Adéquation des Automates Stochastiques Hybrides pour la modélisation des conséquences d'un événement redouté

Carole Duval, Nicolae Brinzei, Hassane Chraibi, Mickaël Hassanaly

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

Adéquation des Automates Stochastiques Hybrides pour la modélisation des conséquences d'un événement redouté

Adequacy of Stochastic Hybrid Automata for the modeling of feared event consequences

Carole DUVAL
EDF-R&D
Palaiseau, France
carole.duval@edf.fr

Nicolae BRINZEI
Centre de Recherche en Automatique de Nancy – CNRS UMR 7039, ENSEM, Université de Lorraine
Vandœuvre-lès-Nancy
nicolae.brinzei@univ-lorraine.fr

Hassane CHRAIBI, Mickaël HASSANALY
EDF-R&D
Palaiseau, France
hassane.chraïbi@edf.fr
mickaël.hassanaly@edf.fr

***Résumé* — Les travaux présentés montrent la faisabilité de la modélisation des conséquences d'un événement redouté avec les Automates Stochastiques Hybrides implémentés dans l'outil d'évaluation de la Sûreté de Fonctionnement (SdF) des systèmes hybrides, PyCATSHOO.**

Mots-clés* — *Analyse de risques, système complexe, infrastructure critique, Automates Stochastiques Hybrides, résilience

Abstract* — *The research work presented in this paper shows the feasibility of the modeling of the feared event consequences by means of Stochastic Hybrid Automata, implemented in the PyCATSHOO Software.

Keywords* — *Risk Analysis, complex system, critical infrastructure, Stochastic Hybrid Automata, resilience

I. INTRODUCTION

Une conception robuste repose sur, d'une part, un dimensionnement réduisant les risques c'est-à-dire incluant les bonnes barrières de prévention de ces risques et doit permettre d'autre part au système d'être résilient c'est-à-dire comporter les barrières de mitigation (ou de protection) pour résister aux évènements redoutés (ER).

Ces deux volets mettent le plus souvent en cause des phénomènes physiques et des comportements aléatoires tels des défaillances et réparations. Ce sont les systèmes dits hybrides pour lesquels des approches spécifiques sont nécessaires. Nous nous intéresserons ici à démontrer la faisabilité de modéliser les conséquences d'un événement redouté sur un système hybride par le formalisme des Automates Stochastiques Hybrides (ASH) [1, 2], rendue aisée par l'outil PyCATSHOO [3, 4] et son couplage avec les équations aux dérivées partielles de convection-diffusion d'un logiciel 3D, *Code_Saturne*, opensource.

II. CONTEXTE

Ces travaux s'inscrivent dans la continuité d'une part, des travaux d'**Analyse intégrée des Risques** qui permettent d'identifier et de quantifier les éléments critiques d'un système socio-technique complexe en lien avec son environnement [5] et d'autre part, du projet européen **FORTRESS** [6]. Sur la base de l'analyse des crises passées sur des infrastructures critiques, ce dernier a mis en évidence l'importance des effets cascades (conséquences de gravité augmentée au travers des infrastructures critiques, exemple de l'ouragan SANDY) et a développé une méthode de modélisation et de quantification des conséquences d'un événement redouté pour en déduire la criticité des différentes parties prenantes. Deux outils y ont été développés pour respectivement modéliser et traiter ces conséquences. **Par une évaluation qualitative à travers de poids affectés à chacune des composantes des conséquences**, le premier outil identifie les nœuds critiques du système et en fournit une hiérarchisation. Cependant, **passer au quantitatif** nous est apparu nécessaire pour donner au décideur des scénarios **quantifiés** de conséquences et surtout prenant en compte les **aspects temporels**, exigence intrinsèque **des conséquences** d'un événement redouté dans lesquelles les **reconfigurations** du système comme les nouvelles défaillances ou leurs récupérations sont à prendre en compte.

Pour répondre à ces exigences, le recours aux Automates Stochastiques Hybrides (ASH) est une solution adéquate et sur cette base, l'outil **PyCATSHOO** permet d'y répondre. Développé depuis 2013 par EDF-R&D et accessible en C++ et en Python, il met en œuvre une version distribuée **des ASH** ; les Automates Stochastiques Hybrides Distribués (ASHD). Cette approche permet de modéliser de manière individuelle les comportements fonctionnels et dysfonctionnels des différents composants. Le résultat est

une bibliothèque de modèles qui permet, par composition, de produire un modèle du système cible dans sa globalité.

Des paramètres liés à la simulation à effectuer sont à renseigner dans un fichier au format XML qui permettent d'affecter par exemple des taux de transition aux automates instanciés avant le lancement de la simulation de Monte-Carlo, elle aussi paramétrable depuis ce même fichier. Il est aussi possible d'isoler les séquences intéressantes pour l'utilisateur et d'afficher l'évolution de grandeurs de la sûreté de fonctionnement [7].

L'intégration de l'aspect continu se fait par des équations différentielles ordinaires (EDO) permettant de modéliser l'évolution d'une conséquence ponctuelle en fonction du temps. Afin de bénéficier de grandeurs physiques plus précises (3D), ces EDO devaient être remplacées par l'appel à un code de simulation permettant d'obtenir l'évolution de variables physiques en fonction du temps en tout point de l'espace du système étudié. **L'objectif est ainsi de montrer la faisabilité d'un tel couplage pour modéliser et simuler les conséquences d'un événement redouté et d'ajouter à cette quantification dynamique hybride la prise en compte de la dynamique des barrières de mitigation, à modéliser dans PyCATSHOO et qui ont un impact sur des conditions aux limites du calcul physique 3D.**

III. METHODE

Après avoir décrit la complexité d'un cas d'application, la méthode de développement retenue pour le traitement des conséquences d'un événement redouté est identifiée et ses différentes étapes présentées : l'identification de la géométrie, la modélisation des différentes parties prenantes et leurs interrelations dans l'outil d'évaluation de la SdF puis le couplage avec le solveur physique.

A. *Logique de développement retenue à partir d'un cas d'application*

On considère un site de stockage de matières dangereuses dont la défaillance d'un des 4 silos conduirait à un nuage toxique. Surélevés de 4 mètres, ils mesurent 4 mètres de côté et 8 mètres de haut. Chacun comporte 4 organes appelés « bypass » qui, défaillants, libèrent du polluant suivant une loi de dispersion en fonction du temps qui diffère suivant le bypass et le nombre de bypass simultanément en panne. La propagation de la défaillance totale d'un silo (4 bypass ouverts) à ses voisins les plus proches survient au bout de 1000s, tous les bypass de ces silos voisins tombent alors en panne. Le site comporte aussi 3 bâtiments considérés comme inaccessibles aux intervenants (ils devront les contourner pour aller effectuer leurs réparations). Enfin, les lois de dispersion précédemment mentionnées sont des conditions limites (CLs) de vitesse et de température décrites dans le Tableau 1.

	Vitesse	Température
Byp. 1	$v_{aerosol} = \begin{cases} 6 - \frac{t_{écoulé}}{1600} \text{ si } t_{écoulé} < 3200\ s \\ 4\ m.s^{-1} \text{ sinon} \end{cases}$	$\theta = 100°C$
Byp. 2	$v_{aerosol} = \begin{cases} 12\ m.s^{-1} \text{ si ouverture totale} \\ 7\ m.s^{-1} \text{ sinon} \end{cases}$	$\theta = \begin{cases} 100°C \text{ si ouverture totale} \\ 130°C \text{ sinon} \end{cases}$
Byp. 3	$v_{aerosol} = 4\ m.s^{-1}$	$\theta = 100°C$
Byp. 4	$v_{aerosol} = \begin{cases} 2 - \frac{t_{écoulé}}{2000} \text{ si } t_{écoulé} < 2000\ s \\ 1\ m.s^{-1} \text{ sinon} \end{cases}$	$\theta = 100°C$

Tableau 1 : Conditions aux limites en vitesse et température pour chacun des bypass d'un silo

Pour garantir la transposition de ce cas à d'autres, la méthode développée est modulaire, sans adhérence aux dimensions et à la configuration du système (nombre de silos, emplacements des bâtiments, nombres de bypass et d'intervenants, de barrières, CLs et maillage) tout en garantissant la cohérence entre les modélisations PyCATSHOO et *Code_Saturne*.

La complexité du problème amène à le décomposer selon les étapes suivantes : l'implémentation de la géométrie dans l'outil d'évaluation de la SdF, la modélisation des différentes parties prenantes de notre système en adoptant le formalisme des Automates Stochastiques Hybrides et leurs interrelations, puis le couplage entre l'évaluation de SdF et le solveur physique.

B. *Introduction de la géométrie*

a) Discrétisation et sectorisation

Essentiellement continue, la **géométrie** est **discrétisée** pour pouvoir calculer sur le maillage associé les grandeurs caractéristiques des conséquences de l'ER (ici les concentrations de polluant) et y appliquer les algorithmes d'optimisation de chemin. Cohérent avec le formalisme discret des ASH, cette **discrétisation de la géométrie se fait dans PyCATSHOO par un processus de sectorisation** qui vise à découper le site visé en une multitude de secteurs élémentaires dans lesquels vont évoluer nos différentes parties prenantes. Au fil des défaillances et réparations, et au gré de la dispersion des nuages polluants, chacun de ces secteurs verra sa concentration en polluant évoluer tout au long de la simulation.

Cette discrétisation diffère a priori du maillage effectué pour le calcul physique qui n'a pas forcément le même degré de finesse, dans le **but essentiel de réduire temps calcul et espace de stockage**. Si besoin, une étape d'interpolation permettra de fournir les variables aux localisations requises.

b) Formalisme des graphes à liens

On lui associe un graphe à liens pour définir l'ensemble des chemins possibles pour un intervenant allant d'un point A à un point B. On rappelle ici ce formalisme : un graphe est un ensemble de nœuds liés par des arêtes.

Plus précisément, un graphe G est un couple (V,E) où V est un ensemble fini de nœuds V = {1,…,n} et E un ensemble de arêtes tel que E ⊆ V x V. Ce graphe G est considéré dépourvu de boucles (∀i ∈ V, (i, i) ∉E) et est non dirigé (∀i, j ∈ V, (i, j) ∈ E ⇔ (j, i) ∈ E) [8]. On se donne pour exemple le graphe suivant :

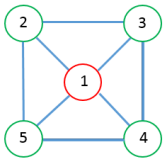

Figure 1 : Un graphe simple

Pour ce graphe G on a alors :

- V = {1, 2, 3, 4, 5}
- E = { (1,2), (1,3), (1,4), (1,5), (2,1), (2,3), (2,5), (3,1), (3,2), (3,4), (4,1), (4,3), (4,5), (5,1), (5,2), (5,4) }

L'implémentation du graphe à liens sous Python peut se faire selon deux formalismes : la liste d'adjacence ou la

matrice d'adjacence. La liste d'adjacence peut prendre la forme d'un dictionnaire sous Python où, pour chaque nœud, on aurait la liste des nœuds auquel il est lié. Ainsi si on reprend l'exemple du graphe précédent, on aurait le dictionnaire suivant :

G = {1 : 2,3,4,5 ; 2 : 1,3,5 ; 3 : 1,2,4 ; 4 : 1,3,5 ; 5 : 1,2,4}

Cette représentation compacte est surtout appropriée à la représentation de graphes de faible densité mais est beaucoup moins intéressante pour des graphes denses comportant un grand nombre d'arêtes. De plus, elle peut s'avérer plus compliquée à manipuler en particulier lorsqu'il s'agit de pondérer les arêtes du graphe que l'on souhaite représenter, alors que la matrice d'adjacence permet de combiner aisément et naturellement les informations d'emplacement et de valeurs de chacun de nos arêtes. Une matrice d'adjacence d'un graphe non-orienté est définie comme une matrice carrée symétrique A = (a_{ij}) pour tout i et pour tout j appartenant à V, de dimension n x n avec n le nombre de nœuds du graphe, avec pour a_{ij} :

$$a_{ij} = \begin{cases} 1 \; si \; (i,j) \in E, \\ 0 \; sinon \end{cases}$$

Dans le cas du graphe simple en exemple, on obtient :

$$A = \begin{array}{c|ccccc} i\backslash j & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 0 & 1 \\ 3 & 1 & 1 & 0 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

La génération et le parcours d'une telle matrice sont beaucoup plus aisés que pour une liste d'adjacence, et les concentrations variables des secteurs, beaucoup plus visibles et accessibles avec cette représentation puisque les concentrations sont reportées directement après calcul sur la valeur des arcs. Pour ces raisons, cette représentation est celle retenue dans le développement.

c) Algorithmes d'optimisation des chemins

Sur cette dernière carte (sous la forme de graphe à liens) le chemin à suivre par un intervenant est optimisé pour minimiser la concentration de polluant rencontrée. L'**algorithme de Dijkstra** est rappelé ici avant d'être implémenté dans le chapitre IV.B.

L'algorithme de Dijkstra, décrit en 1956 et publié trois ans plus tard, est devenu un incontournable de la question du « pathfinding ». Comme l'algorithme de Bellman-Ford, il garantit l'obtention d'une solution optimale sous couvert de remplir les conditions d'application de l'algorithme. Deux variantes de cet algorithme existent, l'une permettant de générer l'arbre de tous les chemins les plus courts depuis un nœud P, et l'autre **donnant le chemin le plus court entre deux nœuds P et Q**. Cette dernière variante est retenue car la première est adaptée à des coûts fixes ou très peu changeants, les deux chemins calculés entre deux appels étant alors très probablement identiques et il est plus intelligent d'effectuer le calcul une seule fois au premier appel et d'avoir tous les chemins possibles mémorisés. L'**environnement étant dans le cas des conséquences d'un événement redouté très évolutif** (concentrations), l'arbre calculé devient rapidement caduc dans la simulation.

Tel que décrit à l'origine, avec P et Q les nœuds respectivement de départ et d'arrivée, on se donne trois ensembles de nœuds :

- un premier ensemble (A1) contenant, dans l'ordre croissant des coûts, les nœuds, pour lesquels le chemin le plus court pour les atteindre depuis le nœud P est connu ;
- un deuxième (A2) qui contient tous les nœuds voisins de ceux placés dans l'ensemble précédemment décrit ;
- un troisième ensemble (A3) qui contient tous les autres nœuds.

Les arêtes sont elles aussi divisées en trois groupes :

- celles liant P aux nœuds du groupe A1 ;
- celles parmi lesquels un sera placé dans le premier groupe et correspond à une bijection du second ensemble de nœuds ;
- et les autres arêtes soit rejetées, soit non-explorées [9].

L'expression très théorique de l'algorithme a été adaptée et allégée pour être plus facilement mise en œuvre en jouant directement sur les distances que l'on initialise à une valeur infinie pour tous les nœuds non parcourus. On met ensuite à jour les distances des nœuds voisins en ajoutant les poids des arcs que l'on aurait à traverser pour les parcourir. On sélectionne le nœud pour lequel la distance est maintenant la plus courte depuis le nœud de départ. On réitère ce processus jusqu'à atteindre le nœud objectif.

L'**algorithme A*** a aussi été implémenté. Il fonctionne sur le même modèle que l'algorithme précédent, avec toutefois une différence cruciale : l'utilisation d'une heuristique. Cette heuristique oriente la recherche opérationnelle dans une direction choisie, direction obtenue par le calcul d'un score, et permet d'obtenir une solution sous-optimale plus rapidement qu'avec Dijkstra. Ce **score** est la somme du plus petit coût à parcourir pour atteindre le nœud visité depuis le nœud de départ, comme dans Dijkstra, et une estimation de celui restant vers le nœud objectif. La principale difficulté d'implémentation d'un tel algorithme réside donc dans la définition de cette heuristique puisque de celle-ci va dépendre en grande partie la proximité de la solution sous-optimale avec celle optimale garantie par la solution précédente.

C. Modélisation du système PyCATSHOO

a) Rappel sur les Automates Stochastiques Hybrides[10] :

Comme évoqué brièvement plus haut, les Automates Stochastiques Hybrides Distribués introduits par PyCATSHOO ajoutent l'aspect compositionnel aux Automates Stochastiques Hybrides qui sont mis en œuvre localement au sein chaque composant du système.

Les ASH sont eux-mêmes basés sur les Automates à Etats Finis auxquels on a ajouté les équations différentielles représentant les variables d'état, les Automates Stochastiques Hybrides sont définis par le 11-uplet suivante :

$$ASH = (S, E, A_c, X, A, H, F, P, s_0, x_0, P_0)$$

S: *ensemble fini d'états discrets* $\{s^1, \dots, s^m\}$
E: *ensemble fini des évènements stochastiques ou déterministes* $\{e_1, \dots, e_r\}$
X: *ensemble fini des variables continues à évolution temporelle* $\{x_i\}_{i \in [\![1,n]\!]}$
A: *ensemble des arcs de transition de la forme* (s, e_j, G_k, R_k, s')
avec s *et* s' *le départ et la destination de l'arc,* e_j *l'évènement associé,* G_k *la garde sur* x *dans* s, R_k *la fonction reset pour* x *dans* s'.

$A_c : X \times S \to (\mathbb{R}^{n+} \to \mathbb{R})$ *fonction d'activité associant à un élément de* $X \times S$
décrivant l'évolution des variables continues dans chaque état discret
H *est un ensemble fini d'horloges sur* $\mathbb{R}$
$F : H \to (\mathbb{R} \to [0,1])$ *associe une distribution à chaque horloge*
$P = [p_i^l]$ *matrice de distributions de probabilités discrètes avec* p_i^l
une probabilité de transition de s^i *vers* s^l *sachant* e : $p(s^i | s^l, e)$
s_0 *: état initial*
x_0 *: valeur de la variable continue dans* s_0
P_0 *: distribution initiale des probabilités de transition*

Un exemple de comportement global d'un ASH est modélisé dans la Figure 2.

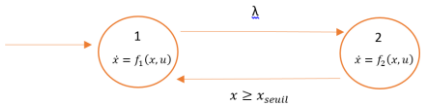

Figure 2 : exemple d'automate stochastique hybride

b) Les Automates Stochastiques Hybrides pour la modélisation des conséquences d'un événement redouté sur un site chimique

En suivant le formalisme précédent, les différents éléments techniques (bypass, silos, choix et comportement des barrières de protection selon le niveau de concentration de polluant) et intervenants ont été modélisés comme suit. Un quartier général (QG) attribue aux pannes les intervenants selon une logique FIFO pour First In First Out, pour dire que le premier arrivé est le premier servi. Le QG n'est pas modélisé par un automate mais par deux files d'attente où les intervenants et les bypasses défaillants sont représentés par des jetons placés dans ces files.

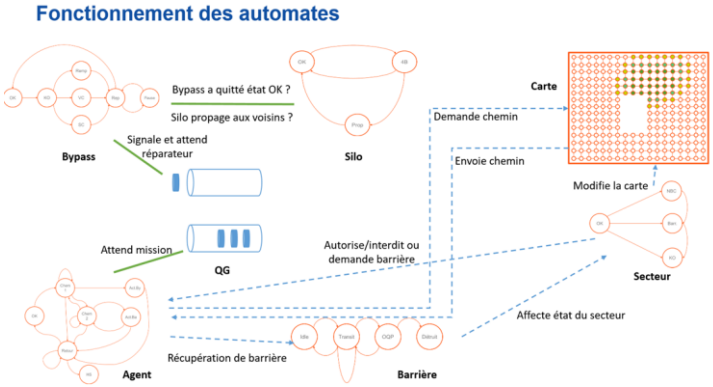

Figure 3 : Automates implémentés dans PyCATSHOO modélisant le bypass, le silo, le comportement de l'intervenant, les barrières de protection

D. *Couplage avec le code de CFD 3D Code_Saturne*

Les phénomènes physiques des systèmes hybrides traités dans PyCATSHOO devant être modélisés en 3D, ses équations différentielles ordinaires sont remplacées par l'appel à un logiciel existant. *Code_Saturne* est un logiciel gratuit et open-source développé par EDF depuis 1997 pour de la CFD (computational fluid dynamics). *Code_Saturne* résout les équations de Navier-Stokes suivantes par une approche en volumes finis pour des flux en 2D, 2D-axisymétrique et en 3D.

$$\frac{\partial \rho}{\partial t} + \mathrm{div}\,(\rho \underline{u}) = \Gamma$$

$$\frac{\partial}{\partial t}(\rho \underline{u}) + \underline{\mathrm{div}}\,(\underline{u} \otimes \rho \underline{u}) = \underline{\mathrm{div}}\,(\underline{\sigma}) + \rho \underline{g} + \underline{ST}_u - \rho \underline{\underline{K}}\,\underline{u} + \Gamma \underline{u}^{in} \quad [11]$$

Il est capable de gérer différents types d'écoulements, incompressibles ou dilatables, avec ou sans transferts de chaleur, grâce à sa prise en compte de tous les types de turbulence, de température et de compressibilité. Il comporte plusieurs modules dédiés à des physiques particulières comme par exemple pour la modélisation de flux atmosphériques, de combustion, de systèmes électriques et de suivi de particules entre autres, le tout utilisable avec une interface graphique ou des scripts Fortran et Python. Le logiciel supporte une grande variété de format pour ses maillages, calcule sur des clusters (des supercalculateurs) et est distribué sur Linux et Windows [12].

La Figure 4 représente la logique d'échange développée dans le couplage, logique asynchrone afin de ne pas contraindre les pas de temps des codes qui pourrait conduire à des difficultés de convergence des outils. Initialement, aucun bypass n'étant tombé, aucun polluant n'est dispersé. Le premier appel se fait alors avec le premier changement de configuration qui est toujours une panne de bypass. Lorsqu'elle survient, les informations d'état des bypasses et silos sont récupérées pour pouvoir fournir à *Code_Saturne* toutes les informations sur les conditions aux limites (CLs) dont il a besoin pour effectuer le calcul de convection/diffusion en plus de celles de vitesse de vent en limite de domaine. Les résultats obtenus sont les concentrations dans les mailles en fonction du temps. Ce fichier de sortie est ensuite interprété par PyCATSHOO pour donner les nouvelles valeurs de concentrations pour chacun des secteurs correspondant (en ajoutant si nécessaire l'étape d'interpolation). Ensuite, lors d'un nouvel évènement affectant la configuration du site comme une nouvelle défaillance de bypass, une pose de barrière ou bien une réparation, soit un nouvel appel s'impose et se traduit dans un nouveau fichier d'entrée avec les différents états des silos et l'emplacement des barrières. Autrement, tant que la séquence simulée par PyCATSHOO n'est pas achevée et si aucun évènement ne survient sur la plage de temps simulée sous *Code_Saturne*, on effectue un nouvel appel au solveur pour la plage de temps suivante.

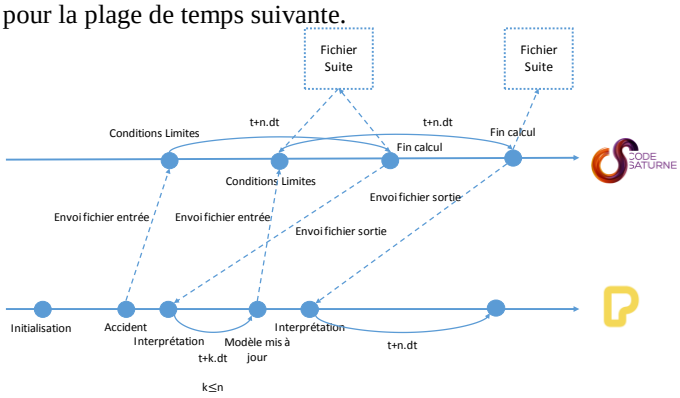

Figure 4 : Séquence retenue de couplage

IV. IMPLEMENTATION

A. *Implémentation de la géométrie*

La géométrie a été décrite sous Salomé [13] qui a permis de générer un maillage de calcul de la convection/diffusion de polluant avec *Code_Saturne*.

Les cotes des objets dans le plan z=0 sont introduites dans un fichier XML (*Struc_Param.xml*) et fournissent les dimensions du site grâce à la variable *v_geom_facility* comprenant la longueur selon l'axe des abscisses et la largeur selon l'axe des ordonnées, le nombre de secteurs selon les mêmes axes avec leur géométrie unitaire, les emplacements de départ des agents (et donc l'emplacement du quartier général (QG)) et la géométrie des bâtiments par le biais des coordonnées relevées précédemment. PyCATSHOO peut alors créer la zone du système étudié et une carte est générée sous la forme de graphe à liens avec des indices interdits (correspondant à des zones obstacles). Une matrice est ainsi obtenue sur laquelle on s'appuie pour les tests d'existence des secteurs, test des (i,j) durant le parcours permettant de définir des zones interdites et sinon, une nouvelle arête liant les deux secteurs vérifiés est ajoutée à la matrice des arêtes du graphe. Cette opération de création des nœuds est répétée dans l'instanciation du système pour générer les objets « secteurs » qui seront liés ensuite aux nœuds qui leur correspondent dans le graphe.

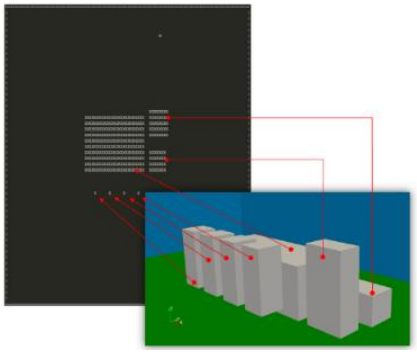

Figure 5 : Affichage de la carte dans la console PyCATSHOO (à gauche) et du site sous Salomé - Paraview (à droite)

B. *Implémentation du cheminement des intervenants*

Les algorithmes de chemin ont été programmés sur la base de cette géométrie en vue de leur affectation à l'intervenant par le QG. Celui de Dijkstra fonctionne bien à l'inverse de A* qui ne fournit pas le chemin de longueur optimale minimisant la concentration rencontrée à chaque nœud. La méthode de passage du chemin généré à une chaîne de caractères permet de vérifier le tracé. Par exemple, pour une défaillance sur le premier silo, la Figure 6 présente le chemin optimal pour un champ de concentration uniforme dans l'espace après 1.05 seconde de calcul.

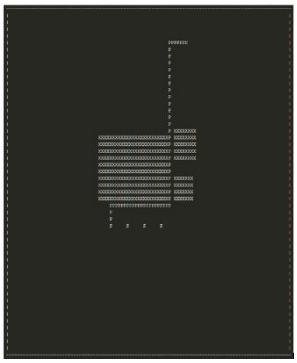

Figure 6 : Tracé du chemin dans la console avec Dijkstra vers le Silo n°1

C. *Implémentation du couplage*

Plusieurs fichiers sont nécessaires au couplage : le fichier XML précédemment cité et généré au préalable avec l'interface graphique *Code_Saturne* constituant la base des manipulations du couplage, deux fichiers textes, l'un comportant les géométries de tous les bypasses et l'autre, les lois suivies par le polluant en fonction de son point d'émission (lois de CLs en fonction du temps), un script python (cs_user_scripts.py) spécifiant notamment le pas du retour arrière dans le calcul. Il est aussi nécessaire de paramétrer les « sondes » qui sont des vecteurs stockant en espace et en temps les informations en concentration depuis le solveur dans un fichier (cs_user_parameters.f90). Une fois placées, elles permettent de générer un fichier (.csv) en fin de calcul qui est ensuite lu par PyCATSHOO pour attribuer aux secteurs leurs concentrations et ce, à chaque pas de temps. Enfin, le lancement du calcul se fait par l'exécution du fichier .bat, écrit et placé dans son répertoire, par PyCATSHOO.

Durant le calcul, lorsque PyCATSHOO fait appel au module de couplage, il lui envoie un vecteur d'information permettant de modéliser l'aggravation du débit au bypass lorsque d'autres sont touchés. Il est décrit comme suit : [[[Booléen testant l'état] pour j compris entre 1 et le nombre de bypasses ; facteur multiplicatif de la vitesse du polluant au Bypass j [pour i compris entre 1 et le nombre de silos]. Par exemple, considérons la situation suivante :

- PyCATSHOO a détecté la défaillance de bypasses B13, B14 pour le silo 1, de bypasses B21, B22, B24 pour le silo 2 et du bypass B31 pour le silo 3,
- avec pour coefficients multiplicatifs de vitesse, 1. si un seul bypass d'un silo est défaillant, 2. si deux bypasses d'un silo sont défaillants, 3 pour trois bypasses défaillants par silo.

Ainsi, onaurait le vecteur suivant relatif aux bypasses :

[[[False, False, True, True], 2],
[[True, True, False, True], 3],
[[True, False, False, False], 1],
[[False, False, False, False], 0]]

Une fois ce vecteur envoyé, si dans le parcours, le module voit une valeur à « True » (c'est-à-dire que le bypass correspondant est défaillant) alors il récupère dans les dictionnaires correspondant les géométries et vitesses de ce bypass et les insère dans le fichier XML qui a été purgé de

façon systématique au début de la procédure d'appel. Ensuite, dans la lecture des arguments (et non plus du vecteur d'information qui faisait lui partie des arguments), le module cherche s'il s'agit d'un « restart » ou non, information elle aussi traduite par un booléen. Le cas échéant, un dernier argument indique l'étape à laquelle reprendre le calcul, information reportée dans le cs_user_scripts.py et l'indicateur de « restart » aussi envoyé dans le XML. Dès lors, le XML est envoyé dans le projet du solveur en plus des fichiers « cs_user_scripts.py » et « cs_user_parameters.f90 » si cela n'a pas encore été fait (ce sera le cas pour les « sondes ») avant d'envoyer le fichier « run_case.bat » et l'exécuter.

Le scénario testé comportait un vecteur d'information réduit sur un seul silo afin de pouvoir tester la conformité du module de couplage. Ce scénario de test était le suivant :

- Défaillance du premier bypass sur une période de 500 secondes ;
- Réparation du premier bypass et défaillance du second bypass sur une période de 500 secondes avec reprise à partir de la défaillance précédente ;
- Réparation du deuxième bypass et défaillance du troisième bypass sur une période de 500 secondes avec reprise à partir de la défaillance précédente ;
- Réparation du troisième bypass et défaillance du quatrième bypass sur une période de 500 secondes avec reprise à partir de la défaillance précédente.

Ce test a donné lieu à une vidéo qui sera présentée en session sur la base des fichiers résultats générés et affichés sous Salomé_Paraview.

Afin de faire fonctionner ce module complètement il reste à tester le bon fonctionnement avec le vecteur d'appel complet depuis une méthode « ode » (Ordinary Differential Equation) dans la simulation de PyCATSHOO et non plus depuis un appel en dur du module. De plus, pendant le test scénario, il était nécessaire de faire patienter la simulation pendant une minute afin de ne pas avoir de conflits sur l'écriture des fichiers résultats. En effet, ceux-ci sont générés et nommés automatiquement par le solveur qui met leur nom au format date-heure à la minute près, ce qui engendre des erreurs critiques avortant les calculs. Il faut donc que dans la procédure d'appel, à la toute fin, le fichier résultat résultant du calcul soit judicieusement renommé pour ne pas avoir ce risque de conflit, ni d'extension inutile du temps de calcul. Mais surtout, l'intérêt de pouvoir effectuer un renommage des fichiers résultats est de pouvoir effectuer des fusions, d'une part, pour réduire la taille des fichiers sortant en écrasant les données inutiles issues des transitoires précédemment calculés finalement écrasés par le dernier appel de calcul, et d'autre part, pour identifier à quelle séquence appartient un fichier résultat en vue de mettre à profit la fonctionnalité de simulations parallèles de PyCATSHOO.

D. Implémentation du comportement du système

Le comportement du système n'a pu être intégralement implémenté. De celui-ci nous avons pour le moment la réception et l'envoi des jetons de missions et d'intervenants depuis le QG et donc tout le processus d'assignation et de mise en attente des missions, le comportement de défaillance et de réparation des bypass ainsi que la procédure d'attente en cas de départ prématuré de l'intervenant, la propagation des défaillances depuis les silos, l'aller des intervenants vers le bypass défaillant et le retour de ces intervenants au QG à la fin de la réparation avec un compteur de concentrations de polluant traversées, la carte qu'ils parcourent avec les secteurs eux aussi générés mais qui nécessitent un fichier contenant des concentrations de polluant à chaque pas de temps pour pouvoir avoir un milieu véritablement changeant.

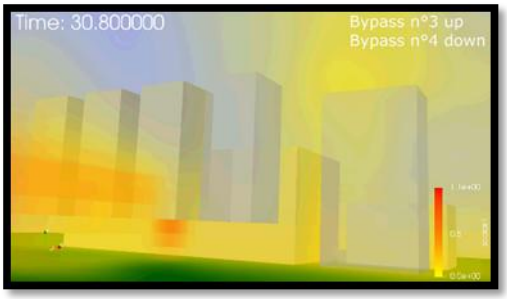

Figure 7 : Capture d'écran de la vidéo résultant du couplage effectué sur la séquence de test

Considérons l'exemple d'une séquence pour voir ce qui se déroule :

Sur une simulation de 10 000s physiques, aucun évènement ne survient dans les 4064 premières secondes. À la 4065^{e}s, une première défaillance apparaît sur le bypass n°4 du silo n°1. Ce bypass signale donc la défaillance au QG qui identifie l'origine et confirme la bonne réception du message. Le QG cherche alors à assigner la mission et le fait pour son premier agent disponible, c'est-à-dire le premier dans sa file d'attente, ici l'intervenant O1. Celui-ci reçoit la mission en même temps que le bypass se voit assigner le jeton de son intervenant. L'intervenant contacte alors la carte pour que celle-ci lui calcule un chemin vers son objectif. L'affichage renvoie alors la chaîne de secteurs à traverser (trop longue pour être affichée ici mais dans la console il suffit de défiler latéralement pour la voir entièrement) ainsi que sa visualisation sur la carte.

```
**************************************************************************
                              SEQUENCE 1
**************************************************************************
[B14    -   4065]   Malfunction detected. Broadcasting alarm.
[HQ     -   4065]   Received alarm from Bypass n°4 on Silo n°1
[B14    -   4065]   Received confirmation from HQ. Waiting for an operative to be assigned...
[HQ     -   4065]   Attempting mission assignment to fix B14...
[HQ     -   4065]   Pool of available operatives: deque(['O.3', 'O.2', 'O.1']) | Alarms received: deque(['B.1.4'])
[HQ     -   4065]   Assigning repair of B14 to O1
[O1     -   4065]   New Mission Received. Received Mission Token: B.1.4
[B14    -   4065]   Operative token received. Waiting for O1 to arrive...
[HQ     -   4065]   Received confirmation from O1. OP has the bypass token.
[HQ     -   4065]   Received confirmation from B14. Bypass has the OP token.
[MAP    -   4065]   Path sent to O1: S|64|46_S|63|46_S|62|46_S|61|46_S|60|46_S|59|46_S|58|46_S|58|45_S|58|44_S|58|4
```

Figure 8 : Premier extrait de la séquence

À la réception du chemin à suivre, l'intervenant détermine le temps estimé avant d'atteindre l'objectif sachant que le franchissement d'un secteur s'effectue toujours dans un temps identique (indiqué en paramètre de la simulation). À l'arrivée, le temps de réparation tiré au préalable est affiché. À la fin de la réparation du bypass B14, aucun autre évènement de défaillance n'est apparu, mais rien ne l'interdit. L'intervenant récupère son jeton et retourne au QG en effectuant le chemin inverse. Au temps nécessaire pour effectuer le parcours s'ajoute un temps de décontamination qui permettra de prendre en compte pour des utilisations futures la proportion de polluant éliminée de

l'intervenant et pouvoir décider s'il est opérationnel ou non. Il s'écoule ensuite plus de 3000 secondes avant la défaillance suivante où c'est O2 qui se retrouve assigné la mission pour aller réparer le bypass n°2 du silo n°4.

Suivant le même processus, on arrive jusqu'au moment de la réparation du bypass défaillant. On ne peut néanmoins voir le reste de l'histoire puisque la simulation est limitée à 10 000 secondes, temps que l'on dépasserait à la fin de la réparation.

```
[O2     -   9553]   Moving to Bypass n°2 of Silo n°4 ETA: 185
[O2     -   9738]   Destination reached. Ready to start repairs.
[B42    -   9738]   O2 has arrived. Received token B.4.2.
[O2     -   9738]   Confirmation from B42 received.
[B42    -   9738]   O2 is ready for repairs. Starting repair process... (estimated time: 2159)
**********************************************************************
```

Figure 10 : Troisième extrait de la séquence

E. Temps de simulation et espace de stockage

Concernant le temps de simulation global, qui inclut le comportement des automates implémentés et des calculs de cheminements, on arrive à des durées **de l'ordre de la minute pour une dizaine de séquences de 10 000 s physiques**. Le nombre de séquences à effectuer pour obtenir des grandeurs de la sûreté de fonctionnement représentatives des phénomènes étudiés varie en fonction du taux de défaillance que l'on se donne.

Quant au temps de calcul pour *Code_Saturne*, sur une machine dotée d'un i7-8750H@2.20GHz avec 32,0 GO de RAM et une nvidia Quadro P1000 de 16 GO de VRAM en soutien, on atteint des temps de calcul de l'ordre de **l'heure pour un calcul sur 500 s physiques**. En moyenne, au vu des logs affichés, il faudrait **effectuer une dizaine de calculs par séquence** : un par panne avec en moyenne deux pannes par séquence survenant vers 5000 secondes pour notre taux de défaillance actuel, un par réparation, un par (re)positionnement de barrière et les continuités de calculs (nécessaires on le rappelle lorsqu'au bout de la période simulée sur *Code_Saturne* aucune modification du système n'a eu lieu et donc un nouveau calcul est appelé pour suivre la progression du nuage sur la plage de temps suivante). Ceci conduit à une dizaine d'heures par séquence sur la machine de développement. Il est ainsi nécessaire de **passer sur un cluster pour ces calculs**.

Ces calculs sont par ailleurs très gourmands en espace de stockage pour le maillage étudié. Les résultats obtenus pour la simulation des 10 séquences précédentes ont occupé 200Go. Cet espace très élevé doit être réduit en stockant moins d'informations **en utilisant des « sondes »**, vecteurs de variables d'intérêt (quitte à procéder à de l'interpolation pour retrouver la connaissance entre ces sondes), plutôt que le stockage systématique de toutes les variables du calcul et le sera encore plus en passant sur le supercalculateur.

V. CONCLUSION

Ces résultats montrent l'intérêt et la capacité de PyCATSHOO à simuler la récupération d'un système complexe suite à un événement redouté, la faisabilité de modéliser et simuler le phénomène physique par un code de calcul 3D externe (*Code_Saturne*) et enfin, d'optimiser les chemins des intervenants durant le processus de réparation.

L'approche de couplage a été établie et les modèles testés séparément. Cette approche et son implémentation informatique sont innovantes car elles permettent d'intégrer une physique tridimensionnelle dans une approche

```
[O1     -   4065]   Moving to Bypass n°4 of Silo n°1 ETA: 275
[O1     -   4340]   Destination reached. Ready to start repairs.
[B14    -   4340]   O1 has arrived. Received token B.1.4.
[O1     -   4340]   Confirmation from B14 received.
[B14    -   4340]   O1 is ready for repairs. Starting repair process... (estimated time: 1807)
[B14    -   6146]   Repairs finished. Giving OP its token back.
[O1     -   6146]   Token O.1 received from Bypass
[B14    -   6146]   Received confirmation from OP. End of procedure. Thank you O1.
[O1     -   6146]   RTB. ETA: 275
[O1     -   6481]   Arrived at base. (S|64|46) Total pollutant taken: 165.0 Transmitted Token. Standing-by.
[HQ     -   6481]   Returning token to the pool. Welcome back O1.
[O1     -   6481]   Received confirmation from HQ. Token in HQ's hands. Standing-by.
[HQ     -   6481]   Attempting mission assignment to our OPS, since O1 is back...
[HQ     -   6481]   No new alarm received yet
[B42    -   9553]   Malfunction detected. Broadcasting alarm.
[HQ     -   9553]   Received alarm from Bypass n°2 on Silo n°4
[B42    -   9553]   Received confirmation from HQ. Waiting for an operative to be assigned...
[HQ     -   9553]   Attempting mission assignment to fix B42...
[HQ     -   9553]   Pool of available operatives: deque(['O.1', 'O.3', 'O.2']) | Alarms received: deque(['B.4.2'])
[HQ     -   9553]   Assigning repair of B42 to O2
[O2     -   9553]   New Mission Received. Received Mission Token: B.4.2
[B42    -   9553]   Operative token received. Waiting for O2 to arrive...
[HQ     -   9553]   Received confirmation from O2. OP has the bypass token.
[HQ     -   9553]   Received confirmation from B42. Bypass has the OP token.
[MAP    -   9553]   Path sent to O2: S|64|46_S|63|46_S|62|46_S|61|46_S|60|46_S|59|46_S|58|46_S|58|45_S|58|44_S|58|
```

Figure 9 : Deuxième extrait de la séquence

d'évaluation de la Sûreté de Fonctionnement simulant les conséquences d'un événement redouté au-delà de sa prévention. Les résultats obtenus prouvent que PyCATSHOO est capable de modéliser, simuler et évaluer ce type de systèmes réels et d'utiliser un solveur pour décrire la physique complexe de convection/diffusion de polluants. Les méthodes théoriques d'optimisation la technique de couplage ont été explorées et les solutions doivent être implémentées.

REFERENCES

[1] J.F. Aubry, N. Brînzei (2015). Systems Dependability Assessment: Modeling with graphs and finite state automata, Wiley-ISTE.

[2] G. Babykina, N. Brînzei, J.F. Aubry, G. Deleuze (2016). "Modeling and simulation of a controlled steam generator in the context of dynamic reliability using a Stochastic Hybrid Automaton", Reliability Engineering and System Safety, 152, 115-136.

[3] PyCATSHOO, http://www.pycatshoo.org/

[4] H. Chraïbi, J.C. Houdebine, A. Sibler, "PyCATSHOO: Toward a new platform dedicated to dynamic reliability assessments of hybrid systems", PSAM 2016, Seoul.

[5] C. Duval, G. Fallet-Fidry, B. Iung, P. Weber and E. Levrat, "A Bayesian network-based integrated risk analysis approach for industrial systems: application to heat sink system and prospects development", Journal of Risk and Reliability, 2012.

[6] FORTRESS, FOResight Tools for REsponding to caScading effectS in crisis, http://cordis.europa.eu/news/rcn/36632_fr.html

[7] H. Chraibi, "Getting started with PyCATSHOO V1.2.2.8 - Document Version V1.1", pycatshoo.org, Palaiseau, 2018.

[8] I.-C. Morarescu, «Collaborative control in multi-agent system,» chez CRAN, Nancy, 2018.

[9] E. W. Dijkstra, «A note on two problems in connexion with graphs,» Numerische Mathematik, vol. 1, pp. 269-271, 1959.

[10] G. A. Castañeda, J.-F. Aubry, N. Brinzei, "Stochastic hybrid automata model for dynamic reliability assessment", Proc. IMechE, vol. 225, pp. Reliability, Part O: J.Risk and, 08 Juillet 2010.

[11] *Code_Saturne* documentation [en ligne], https://www.code-saturne.org/cms/sites/default/files/docs/5.3/theory.pdf · Fichier PDF

[12] EDF, " Description of *Code_Saturne* " [en ligne]. Available: http://www.code-saturne.org/cms/features

[13] Salomé, https://www.salome-platform.org/user-section/about/mesh