Utilisation de modèles métiers pour l'orchestration d'une simulation distribuée fédérant la gestion des risques

Simon Gorecki

THÈSE PRÉSENTÉE
POUR OBTENIR LE GRADE DE

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DES SCIENCES PHYSIQUES ET DE L'INGÉNIEUR
SPECIALITÉ : Automatique, Productique, Signal et Image, Ingénierie cognitique

Par **Simon GORECKI**

UTILISATION DE MODÈLES MÉTIERS POUR L'ORCHESTRATION D'UNE SIMULATION DISTRIBUÉE FÉDÉRANT LA GESTION DES RISQUES

Sous la direction de Yves DUCQ
Co-directeur : Gregory ZACHAREWICZ
Co-encadrant : Nicolas PERRY

Soutenance prévue le lundi 7 décembre 2020

Membres du jury :

M. TRAORÉ Mamadou Kaba Professeur des Universités	Université de Bordeaux	Président
M. BÉNABEN Frédérick Professeur de l'IMT	IMT – Mines Albi	Rapporteur
M. SIRON Pierre Professeur de l'ISAE	ISAE-SUPAERO Toulouse	Rapporteur
M. DUCQ Yves Professeur des Universités	Université de Bordeaux	Directeur
M. ZACHAREWICZ Gregory Professeur de l'IMT	IMT – Mines Ales	Co-directeur
M. PERRY Nicolas Professeur des Universités	Arts et Métiers Bordeaux	Co-encadrant
M. COUDRAIS-DUHAMEL Michel Responsable R&D	ALSOLENTECH	Invité
M. RIBAULT Judicaël Docteur	Université de Bordeaux	Invité

Titre : Utilisation de modèles métiers pour l'orchestration d'une simulation distribuée fédérant la gestion des risques

Résumé : De nos jours, la modélisation et la simulation (M&S) jouent un rôle central dans le dimensionnement des processus métiers de l'industrie en raison d'une augmentation significative des composants qui y participent. La combinaison des composants produit un système complexe et induit une augmentation proportionnelle des risques inhérents aux projets. En conséquence, il est nécessaire de caractériser et maîtriser ces derniers dans les modèles et la simulation. De plus, le système étant l'association de différents domaines, les acteurs doivent pouvoir effectuer des simulations intégrant des composants spécifiques, indépendants, hétérogènes et distribués. Plusieurs standards de co-simulation proposent des mécanismes pour orchestrer des composants distribués. Parmi eux, deux standards : High Level Architecture (HLA) et Functional Mock-up Interface (FMI) permettent l'association de composants hétérogènes et possèdent respectivement des atouts d'expression des comportements et de gestion du temps. Cependant leur mise en œuvre reste compliquée avec des outils de gestion des processus et un besoin de compatibilité est toujours attendu. De plus, les implémentations conjointes des deux standards présentent des problèmes d'alignement des concepts et méthodes pour une orchestration globale des deux standards.

Dans ce travail de recherche, nous proposons le développement d'une plateforme de modélisation et de simulation dans le but d'appréhender de façon plus formalisée la complexité du contexte industriel. Notre objectif est de proposer une méthode et un outil modulaire capable de résoudre différentes problématiques industrielles. Chaque thématique de recherche présentée ici sera traitée par une extension de la plateforme ayant pour rôle de répondre à une problématique spécifique :

- Modélisation et simulation des risques inhérents à un contexte industriel :
  Nous proposons une extension capable d'externaliser la définition de risques contextuels hors d'un modèle de simulation, afin de faciliter le développement et la simulation de ces derniers.
- Interaction/Intégration entre deux normes de co-simulation (HLA et FMI) :
  Nous proposons l'adaptation de notre plateforme de M&S au pilotage, et à la communication des deux standards de co-simulation.
- Modélisation et orchestration d'une simulation distribuée :
  Nous proposons l'utilisation d'un langage graphique intégré à la plateforme pour le pilotage d'une simulation distribuée HLA.

Ces contributions sont opérationnalisées et expérimentées dans le domaine de la production d'usines mobiles à énergies solaires par l'entreprise ALSOLENTECH. En effet, cette société a exprimé des besoins s'intégrant bien dans nos problématiques de recherches, elle nous fournit un cas d'étude et d'application et exploite nos travaux depuis les concepts jusqu'à l'exploitation des solutions proposées.

MOT CLES : Co-simulation, Modélisation, Simulation, HLA, FMI, Gestion des risques

Title: Using business models to orchestrate a distributed simulation federating risk management

Abstract: Nowadays, modeling and simulation (M&S) plays a central role in the dimensioning of business processes in industry due to a significant increase in the number of components involved. The combination of components produces a complex system and induces a proportional increase in project risks. Consequently, it is necessary to characterize and control them in models and simulation. In addition, since the system is the association of different domains, actors must be able to perform simulations integrating specific, independent, heterogeneous, and distributed components. Several co-simulation standards propose mechanisms to orchestrate distributed components. Among them, two standards: High Level Architecture (HLA) and Functional Mock-up Interface (FMI) allow the association of heterogeneous components and respectively have advantages in terms of behavior expression and time management. However, their implementation remains complicated with process management tools and a need for compatibility is always expected. In addition, joint implementations of the two standards present problems in aligning concepts and methods for a global orchestration of the two standards.

In this research work, we propose the development of a modeling and simulation platform to apprehend in a more formalized way the complexity of the industrial context. Our objective is to propose a method and a modular tool capable of solving different industrial problems. Each research topic presented here will be treated by an extension of the platform whose role is to respond to a problem related to a different complexity:

- Modeling and simulation of the risks inherent to an industrial context:
  We propose an extension capable of externalizing the definition of contextual risks outside of a simulation model, to facilitate their development and simulation.
- Interaction/Integration between two co-simulation standards (HLA & FMI):
  We propose the adaptation of our M&S platform to the piloting and communication of the two co-simulation standards.
- Modeling and orchestration of a distributed simulation:
  We propose the use of a graphical language integrated to the platform for the piloting of a distributed HLA simulation.

These contributions are operationalized and experimented in the field of the production of mobile solar energy plants by the company ALSOLENTECH. Indeed, this company has expressed needs that fit well with our research problems, it provides us with a case study and application and exploits our work from the concepts to the exploitation of the proposed solutions

Keywords: Co-simulation, modeling, simulation, HLA, FMI, Risk management

Remerciements

Tout d'abord, je tiens à exprimer mes plus sincères remerciements et ma plus profonde gratitude envers M. Gregory Zacharewicz, M. Yves Ducq et M. Nicolas Perry pour leur soutien sans faille et la bienveillance qu'ils m'ont apporté durant ces années de thèse. Leur implication et leur disponibilité ont été pour moi d'un grand réconfort et une source de motivation. Je ne saurai leur dire combien leur aide m'a été précieuse dans l'aboutissement de ce travail.

J'adresse également ma gratitude à M. Mamadou Kaba Taoré, Professeur à l'Université de Bordeaux pour avoir accepté de présider ce jury, mes respectueuses reconnaissances à M. Frédérick Bénaben, Professeur à l'IMT Mines Albi, et M. Pierre Siron, Professeur à ISAE-SUPAERO Toulouse pour l'intérêt qu'ils ont porté à ce travail en acceptant d'être les rapporteurs de mes travaux de thèse.

J'adresse également mes remerciements et ma sympathie à M. Judicaël Ribault, M. Youssef Bouanan, et M. Michel Coudrais-Duhamel pour leur aide précieuse et leur bienveillance.

Je tiens à exprimer ma sympathie à l'ensemble des membres permanents et temporaires du laboratoire IMS de l'Université de bordeaux. Je pense chaleureusement à mes amis Jalal, Kawtar, Yousra et Safaa pour l'aide, la gentillesse et les bons moments que j'ai passé avec eux.

Enfin, au terme de ce parcours, ma gratitude va à mes parents, Laurent et Catherine, mon frère Martin, mes amis et le reste de ma famille pour le soutien sans faille qu'ils m'ont apporté durant toutes ces années d'études.

Simon Gorecki
Université de Bordeaux
Octobre 2020

Sommaire

Table des illustrations

Table des tableaux

Chapitre 1
Introduction, Enjeux de recherche et structure

Sommaire

1.Introduction générale

L'aire de l'industrie 4.0 est considérée comme la 4ème révolution industrielle. Elle succède à l'arrivée de l'automatisation de la production notamment avec les automates et les robots [1]. Les révolutions industrielles sont des processus de changement rapide de l'industrie qui modifient en profondeur ses activités (voir Tableau 1).

Révolution	Date	Caractéristiques	Production
1er révolution	1780	- Exploitation du charbon - Machine à vapeur	Mécanisée
2ème révolution	1850	- Electricité - Exploitation du pétrole - Technologies de communication (téléphone)	De masse
3ème révolution	1970	- Automates et robots - Informatique industrielle - Energies renouvelables - Internet	Automatisée

Tableau 1 : Historique des différentes révolutions industrielles

Le concept de quatrième révolution industrielle est apparu au début du XXIème siècle. Kagermarin et al. font référence à la numérisation de la production et décrivent la vision d'une usine intelligente, caractérisée par la mise en réseau de tous les processus de production afin de fonctionner de manière interactive et d'augmenter la productivité et l'efficacité de l'utilisation des ressources [2]. Ces avancées peuvent être divisées en quatre piliers qui forment l'industrie du futur :

- La numérisation des usines. Le développement du cloud et du big data permettent une gestion décentralisée de grandes quantités de données. Ces données proviennent du concept d'IOT (où Internet Of Things) qui propose l'exploitation d'objets connectés permettant la récolte de grandes quantités de données sur des utilisateurs et/ou des systèmes industriels. L'interconnexion des objets et la communication instantanée intra-entreprise permettent de faire circuler les informations récoltées, déduites ou anticipées afin de représenter numériquement une version virtuelle de l'usine.
- La flexibilité des usines. Les avancées en matière d'impression 3D, de robotique, de cobotique (collaboration homme-robot), de drones, et d'Intelligence Artificielle permettent une plus grande flexibilité de production, ce qui améliore les conditions de travail, tout en contribuant à la compétitivité de l'entreprise.
- Les nouveaux outils logistiques. L'enjeu de l'usine du futur consiste donc à être dirigée par les données. La maîtrise des processus et des informations amène au contrôle et à la réduction des risques. Le cycle numérique doit parfaitement intégrer les processus de l'entreprise comme la gestion clients (CRM : Customer Relationship Management), la gestion des entrepôts (WMS : Warehouse Management System) ou la gestion de la supply chain

(SCM : Supply Chain Management) au sein d'un ERP (Entreprise Ressources Planning) capable d'intégrer tous les flux.

- Les interactions homme-machine (IHM). Les interactions IHM sont déployées dans tous les secteurs d'activité où l'utilisation de machines ou d'appareils automatiques nécessite une intervention humaine. Le besoin d'efficacité, de mobilité, de services à distance et de matériel fiable pousse l'industrie à faire preuve d'innovation pour faire évoluer ses techniques (réalité augmentée, réalité virtuelle, objets connectes, etc.).

D'autre part, comme le soulignent A. Adamik et al., dans [3], les entreprises sont confrontées à une concurrence croissante sur leurs marchés qui pousse les gestionnaires à constamment rechercher de nouvelles méthodes, approches et stratégies pour améliorer la flexibilité opérationnelle, l'efficacité, l'innovation et la réactivité grâce à un contrôle précis soutenu par les technologies 4.0 [4]. L'utilisation d'approches numériques anticipe le développement et le contrôle des systèmes de production intelligents en contribuant, par exemple, à la maîtrise du comportement des flux de production. Dans le contexte de l'industrie 4.0, le besoin de modélisation et de simulation (M&S) devient de plus en plus important [5] car il permet aux utilisateurs de manipuler des prototypes virtuels pour reproduire et renouveler facilement les expériences dans différentes situations tout en travaillant sur des modèles virtuels accessibles. Cette approche par la modélisation et la simulation entre en lien avec les quatre piliers énoncés de l'industrie 4.0 : (1) L'interconnexion des systèmes et la gestion de grandes quantités de données permettent l'enrichissement des données d'entrée d'un modèle ; (2) La simulation d'un processus industriel permet une représentation et une analyse efficace de son fonctionnement. Il en découle une capacité de prise de décision et d'adaptation à son environnement accru : (3) Un outil de M&S peut être connecté à un système d'information pour entreprise en tant que module d'aide à la décision ; (4) Comme dans toutes opérations impliquant l'homme et une machine, le besoin d'efficacité et de simplicité d'utilisation sont nécessaires. C'est également le cas pour les outils de modélisation et simulation.

Le modèle commercial de l'industrie 4.0 s'appuie sur les données pour maîtriser les processus. L'utilisation d'informations récoltées ou prédites permettent d'analyser et de comprendre des systèmes complexes et d'en maîtriser leurs fonctionnements. La capacité à virtualiser des espaces de travail, ou des systèmes permet de combler le fossé entre le monde physique piloté par des machines, et le monde virtuel. Selon Madni et al. [6], le Digital Twin (DT) crée l'environnement parfait pour la collecte de données sur tous les aspects du processus de fabrication à des fins d'analyse et de simulation. Lorsque les données sont collectées avec précision et qu'un Digital Twin est conçu, les intégrateurs de systèmes, les analystes de données, et les autres acteurs peuvent l'utiliser pour mener des politiques commerciales et améliorer les processus décisionnels. Selon [7], d'ici 2021, près de la moitié des grandes entreprises industrielles utiliseront le DT pour faciliter l'évaluation des performances et des risques techniques de systèmes industriels.

Cependant, les systèmes devenant de plus en plus volumineux et complexes, la difficulté de simulation de ces systèmes et les risques liés augmentent

proportionnellement. Comme le décrivent Taylor et al., lorsque le modèle est complexe et nécessite des traitements de ressources hautes performances, l'approche de simulation classique devient insuffisante ; l'exécution du modèle doit être divisée et répartie entre un grand nombre de processeurs ou de machines, sur la base d'outils de modélisation appropriés, de manière intégrée [8]. Cette approche de modélisation et de simulation doit prendre en compte à la fois la notion de distribution des complexités, mais aussi la gestion les risques afin de sécuriser la mise en œuvre et l'exécution du système simulé proposé [9].

Pour répondre à ce besoin, les mécanismes permettant l'interopérabilité entre les composants de simulation hétérogènes doivent être favorisés afin de permettre la réutilisation des composants, la factorisation des complexités, et la réduction des coûts de développement.

D'après les éléments énoncés de cette sous-partie, le comportement d'un système peut être anticipé en modélisant et en simulant son comportement dans un environnement soumis à des incertitudes. Un processus d'entreprise nécessite l'expertise de plusieurs domaines, il est donc représenté par un scénario qui interconnecte des blocs de fonctionnalités ainsi que des interfaces assurant une connexion avec l'environnement extérieur. Dans le but d'obtenir un outil d'aide à la décision flexible, ces simulations doivent pouvoir s'appuyer sur des composants, et des sources de données hétérogènes.

2.Contexte industriel

Du 30 novembre au 11 décembre 2015 a eu lieu à Paris la COP21 [10], une conférence internationale sur le climat qui a réuni plus de 150 pays. Les enjeux de cet événement étaient de trouver des accords internationaux pour contenir le réchauffement climatique en dessous des 2°C via des mesures politiques et économiques. Parmi ces décisions, les principales s'articulent autour de la réduction des émissions de gaz à effet de serre, ou la favorisation du développement des énergies renouvelables. Dans une démarche similaire, Horizon 2020 [11] (ou H2020) est un programme de la comission européenne de recherche et développement. Avec un budget de 79 milliards d'euros, il a pour objectif de favoriser l'excellence scientifique, la primauté industrielle et les défis sociétaux des différents pays membres de l'UE. Parmi les défis sociétaux, on peut voir apparaitre les problématiques en enjeux en lien avec les changements climatiques ou la transition énergétique.

Ces travaux de thèse s'inscrivent dans le cadre d'un projet innovant évoqué lors de la COP21 et H2020 : l'exploitation de l'énergie solaire, en collaboration avec la société ALSOLENTECH [12]. ALSOLENTECH est une filiale d'ALSOLEN, qui bénéficie d'un actionnariat stable entre le groupe ALCEN : industriel français de hautes technologies, constitué de 27 filiales, et MASEN (Moroccan Agency for Sustainable Energy) : Agence gouvernementale Marocaine en charge du pilotage du développement des énergies renouvelables du Pays.

ALSOLENTECH est une société française basée à Mérignac, proche de Bordeaux, ayant pour activités : la conception, la réalisation et l'exploitation de centrales solaires thermodynamiques et photovoltaïques. La technologie de centrales proposée par l'entreprise permet d'alimenter des zones isolées simultanément en électricité, eau potable, froid, et chaleur industrielle.

Les recherches présentées dans ce manuscrit sont réalisées dans le cadre du projet Region Nouvelle Aquitaine, DIAMANTR qui est le volet recherche du projet DIAMANT. Ce dernier projet a fourni un outil au projet SolR², qui a eu pour objectif de concevoir une usine mobile capable de se déployer dans une zone ensoleillée, et de fabriquer des champs solaires en utilisant la technologie des miroirs de Fresnel. Devant ce défi, plusieurs partenaires se sont associés au projet pour le mener à bien :

- L'Université de Bordeaux.
- Le CEA Tech (Commissariat à l'Energie Atomique et aux énergies alternatives) qui a contribué avec l'Université à la création d'un outil adapté aux besoin d'ALSOLENTECH.
- La région Nouvelle Aquitaine qui a financé une partie du projet.

Afin de faciliter le développement de SolR², ALSOLENTECH, l'Université de Bordeaux et le CEA Tech ont collaboré dans le but de concevoir un outil baptisé DIAMANTR, répondant à des problématiques de recherches, et dont les applications correspondent aux besoins métiers de l'entreprise. DIAMANTR est un outil de modélisation et de simulation de processus métiers. Il a pour vocation de représenter le système d'usine mobile dans sa globalité en incluant la simulation des sous-processus de l'entreprise, tout en prenant en charge la gestion des aléas, et les interactions avec l'environnement du système. Son fonctionnement se veut simple d'utilisation et capable de simuler ou de piloter des composants hétérogènes externes. L'objectif de l'outil est de faciliter le développement du projet SolR², mais aussi les autres projets de l'entreprise, comme la centrale solaire thermodynamique à concentration, ou de façon plus globale, des projets nécessitant la modélisation, la simulation, et le pilotage de processus hétérogènes soumis à des aléas.

3.Problématiques de recherche

Les problématiques de recherches de ce manuscrit seront traitées sous la forme de questionnements scientifiques. D'abord d'un point de vue conceptuel, puis décliné techniquement dans la suite du manuscrit.

Le contexte technique et économique de la production énergétique à concentration de chaleur est un domaine de recherche relativement jeune et qui se heurte à des notions d'incertitude et de compétitivité. De nombreuses entreprises utilisent des outils pour gérer la production et assurer le meilleur rapport qualité / prix, respecter le délai de production de leurs produits et/ou de leurs services [13]. Dans un contexte incertain, il est d'autant plus important d'avoir recours à des outils de modélisation et de simulation afin de virtualiser les systèmes et processus métiers

de façon à étudier et à anticiper leurs comportements. Tout projet, quel que soit son domaine, est soumis durant son cycle de vie à de nombreux risques et aléas provenant d'origine internes ou externes (risques naturels, techniques, politiques, économiques, etc.). La caractérisation et la gestion de ces risques est cruciale pour le bon déroulement du projet. Beaucoup d'outils de gestion des risques existent sur le marché, mais ils ont la particularité de lier étroitement la modélisation des aléas, et les processus qu'ils ciblent. Un projet étant généralement associé à des risques, il est cohérent de traiter conjointement la modélisation des processus et la modélisation des risques. Cette démarche de définition de processus est cohérente, mais implique :

- Une complexification des modèles : celle-ci augmente le nombre d'informations dans le modèle simulé et le rend plus difficilement interprétable par un utilisateur.
- Une faible modularité : les risques métiers et techniques étant associés à un processus, ils ne peuvent pas être appliqués à un autre modèle.

Une première piste de recherche consiste à diviser les données d'entrée d'une simulation en deux : d'un côté un modèle de processus industriel avec un jeu de données d'entrée nominales, et de l'autre un modèle de description des risques et aléas impactant le processus simulé. On peut formuler cette problématique sous la forme suivante :

Q1 : ***Dans un contexte de simulation industrielle, comment augmenter la flexibilité de la M&S, notamment pour prendre en compte la gestion des risques et des aléas de façon modulaire.***

Une industrie est par nature un regroupement de domaines et de technologies variés dont l'imbrication permet la valorisation de produits ou de services. L'ensemble des acteurs d'une chaine industrielle regroupe des outils qui seront amenés à communiquer sans pour autant prévoir d'interface d'échanges normalisés. Dans le cadre de l'industrie 4.0, deux des principaux enjeux passent par la Modélisation et la Simulation (M&S) des systèmes, pour étudier et prévoir leurs comportements au sein d'un environnement (DT), et le dialogue entre les différents éléments de la production (IOT). Ce besoin de simulation implique des problématiques d'interopérabilité entre composants hétérogènes qui sont abordées par les concepts de simulation distribuées : chaque composant représente une sous-partie d'un système global pour expérimenter et analyser les performances de l'ensemble du système. Il existe sur le marché différents standards de simulation distribuée, chacun avec des forces et des faiblesses. De plus, il existe également une faible capacité de communication entre les différents standards. Une des pistes de recherche de ce manuscrit consiste à identifier les principaux standards de co-simulation, et proposer une interface de communication entre eux, ainsi qu'avec leurs environnements.

Q2 : ***Comment créer une interface de communication entre plusieurs outils de M&S compatibles avec des standards de co-simulation ?***

La simulation distribuée représente une discipline et une approche efficace pour faire face à la complexité croissante de l'analyse et de la conception des systèmes modernes et des systèmes de systèmes. Cependant, le développement et l'utilisation de modèles de simulation restent des tâches difficiles et nécessitent un

effort de développement considérable qui se traduit souvent non seulement par une augmentation du temps de développement, mais aussi par une relative fiabilité des résultats. On observe un besoin de simplification tant du côté des développeurs que du côté des utilisateurs. En parallèle au besoin de simplification, le contexte industriel de ce manuscrit exprime le besoin de pilotage d'un ensemble d'outil de simulation. En effet, l'administration d'une simulation par un composant orchestrateur permettrait le pilotage d'un ensemble de sous-composants. Cela peut se traduire par la nécessité d'un modèle décrivant un processus métier, faisant référence à des sous-composants représentant des processus fonctionnels ou techniques. On peut donc en déduire l'interrogation suivante.
Q3 : ***Comment définir un composant orchestrateur de plusieurs sous composants hétérogènes de M&S ?***

4.Structure du manuscrit

La suite de ce manuscrit est organisée de la façon suivante. Le chapitre 2 aborde les notions générales à propos des pratiques et méthodes de gestion de projet pour identifier la notion de risque. Après un bref état de l'art sur les concepts de modélisation et de simulation, nous balayerons les différents outils permettant la gestion des risques dans un contexte industriel avec pour but d'identifier un support d'implémentation technique. Enfin, nous aborderons les différents outils existants dans la littérature pour résoudre les problématiques d'interopérabilité de simulations.

Nous consacrerons ensuite trois chapitres pour répondre aux questions des problématiques évoquées précédemment. Le chapitre 3 présente la contribution concernant l'augmentation de la flexibilité de modélisation, et la gestion des risques et aléas dans un outil de modélisation et de simulation. Le chapitre 4 se consacre aux interfaces de communication entre outils et standards de co-simulation. Dans le chapitre 5, nous présentons la contribution relative à l'orchestration de sous composants hétérogènes. Enfin, le chapitre 6 présentera un cas d'étude validant les 3 contributions de ce manuscrit, pour finir sur une conclusion et les perspectives de ce travail de recherche.

Chapitre 2
Etat de l'art

Sommaire

1.Pratiques et méthodes de gestion de projet

1.1.Introduction

De nos jours, dans le contexte actuel du marché et dans un souci de performances, les entreprises doivent offrir des produits, novateurs et à moindre coût dans des délais courts. Dans le but d'anticiper et corriger les erreurs de conception, une attention particulière est accordée aux outils et méthodes de modélisation et simulation. De plus, chaque projet, parce qu'il est généralement novateur, est soumis à de nombreux risques. Être capable de les maîtriser au mieux est un point crucial dans un contexte incertain. Comme présenté dans le chapitre précédent, un des objectifs de ce travail de recherche est de proposer des méthodes permettant de gérer simultanément et de façon claire la modélisation et la simulation des risques dans un contexte de projet industriel instable.

Un projet est décrit comme un ensemble de ressources collaborant et évoluant dans un environnement en vue d'atteindre des objectifs définis, tout en respectant des contraintes comme les coûts, le temps etc. À ce titre, il répond à la définition d'un système sur lequel il est possible d'obtenir une vue externe, décrivant l'environnement avec lequel il est en interaction et une vue interne permettant d'expliciter sa composition et son organisation, faisant apparaître les éléments constitutifs [14].

De fait, le projet est un ensemble constitué d'un grand nombre d'entités en interaction empêchant l'observateur de prévoir aisément son comportement ou son évolution : un système complexe soumis à l'incertitude.

La complexité traduit la difficulté, voir l'impossibilité, d'appréhension, d'analyse, de maîtrise, et d'anticipation [15]. Elle se manifeste par l'apparition de phénomènes qui, d'après les connaissances sur les comportements et interactions des éléments, ne sont à priori pas prévisibles. D'autres manifestations de la complexité inhérente au système peuvent être définies [16] comme :

- L'aspect dynamique des interactions au sein du système ;
- L'existence de nombreux liens de causalité circulaire et de récursivité dont l'enchevêtrement est tel que les phénomènes en deviennent incompréhensibles et incontrôlables ;
- L'inhérence de l'indécidabilité, de la difficulté de prendre des décisions ;
- L'instabilité et l'évolution par bifurcations causant des changements d'état soudains ;
- La coexistence de logiques différentes, si ce n'est antagonistes.

La deuxième conséquence clef identifiée est celle de l'incertitude, qui impacte directement le projet et son environnement.

Une approche systémique du projet permet de mieux percevoir, et de mieux comprendre le projet dans toute sa complexité, et de s'adapter au caractère dynamique et évolutif du projet [17], [18].

La notion de projet et la gestion qui lui est rattachée existe et se développe depuis la fin du XXème siècle avec l'accroissement de la compétitivité inter-entreprises et une complexification de leur organisation [19]. Face à cette évolution, les entreprises ont été obligées de proposer de nouveaux services et produits innovants, de qualité accrue et à des coûts toujours plus bas. Ainsi, le projet est perçu comme une organisation temporaire évoluant dans l'espace et le temps avec un objectif. C'est un système, composé de sous-systèmes interagissant les uns avec les autres, et avec l'environnement, dans un but global.

Dans le but de prendre en compte l'incertitude et la gestion des risques d'un système complexe, il est impératif de passer par des phases de modélisation et de simulation. En effet, on ne résout pas un problème directement sur un système réel, mais toujours au travers d'un modèle que l'on construit, qui représente ce système. Un modèle est donc une représentation d'un système, il est construit pour permettre de trouver une réponse à une problématique ciblée. En agissant sur les modèles à partir de jeux de données et une base de temps, on définit une simulation.

Les outils de modélisation et simulation ont été développés pour répondre à la nécessité de maîtriser les systèmes complexes. Face aux nombreux défis et à la multiplication des contraintes, l'ingénierie système [20] est une approche intéressante pour le développement et la vérification d'une solution.

Ce chapitre propose un état de l'art des différents domaines liés à notre contexte de recherche. Dans un premier temps, nous définirons la gestion des risques relatifs à un projet. Afin de gérer ces risques, nous présenterons les méthodes de modélisation et simulation, ainsi que les outils implémentant ces concepts. Par la suite, nous nous intéresserons aux problématiques d'interopérabilités notamment au sein des outil de M&S. Enfin, nous développerons les concepts et les principes de fonctionnement des simulation distribués ainsi que de la co-simulation.

1.2.Gestion des risques

Avant de considérer la gestion des risques dans le cadre d'une simulation, il est nécessaire de se poser la question : « qu'est-ce-que le risque ? ».
Le terme de risque est souvent associé au danger, à une situation dangereuse ou un accident. Il existe de nombreux termes souvent mal définis, difficiles à différencier les uns des autres, tant de nombreux écrits confondent le danger et le risque. De plus, le terme « risque » peut être considéré comme la conséquence ou la cause d'un autre évènement [21].
Suivant notre contexte, nous nous intéressons plus particulièrement à la notion de « risque industriel » qui sont susceptibles d'inclure des composantes techniques, et économiques [22] :

- Rendre difficile la compréhension des risques et leurs conséquences dans leur globalité

- Subjectiver lorsque ce dernier se focalise sur une de ces composantes exclusivement.

Dans un contexte industriel, le risque est fonction des critères de :

- Probabilité d'occurrence des phénomènes dangereux
- D'intensité des effets de ces phénomènes qui doivent être estimés ou connus par l'équipe projet.

Dans la littérature, on trouve quasiment autant de définitions du risque, que d'auteurs. La notion de risque est fortement polysémique et a donc été l'objet d'un grand nombre de définitions [23]. Une définition qui peut être considérée comme générale est celle donnée par la norme ISO 31000 dans le cadre de la gestion de projet : « *Effet de l'incertitude sur l'atteinte des objectifs* » [24].
Cette définition met en avant deux notions :

- L'incertitude est définie comme « l'état, même partiel, de défaut d'information concernant la compréhension ou la connaissance d'un évènement, de ses conséquences ou de sa vraisemblance » [25].
  Le caractère innovant d'un projet, le manque d'expérience, un nombre trop élevé de facteurs dépendants entre eux, et la possibilité d'occurrence d'évènements, introduit dans un contexte donné la notion d'incertitude [26].
- Un projet, caractérisé par la complexité, est soumis à l'incertitude. On peut le définir comme un système dont le comportement est difficile à prévoir. Le projet est influencé tout au long de son cycle de vie par différents évènements internes ou externes susceptibles de modifier son déroulement. Ces perturbations peuvent par exemple passer par un allongement des délais, une augmentation des coûts, ou encore une qualité produite moindre. La notion de risque projet correspond alors au panel de ces perturbations pouvant survenir au cours d'un projet [27].

Conformément à la définition fournie par la norme ISO 31000 [24], nous pouvons introduire un certain nombre de concepts gravitant autour du risque et conditionnés par son environnement et ses propres actions. Le risque est alors perçu comme la conséquence d'évènements, d'origine interne ou externe susceptibles d'affecter l'obtention de l'objectif initialement fixé, c'est à partir de cette définition que nous définirons les concepts à représenter plus tard.
Cette définition met en exergue deux aspects fondamentaux du risque, à savoir la probabilité d'occurrence, et l'impact, positif ou négatif, qui définit l'importance des perturbations occasionnées par l'occurrence du risque. Trois types d'impacts sont directement reliés aux objectifs du projet [18] :

- Impacts sur le temps : dérive positive ou négative des délais ;
- Impacts sur les coûts : économies ou surcoût du projet ;
- Impacts sur la qualité ou les performances : amélioration ou dégradation du mode de fonctionnement nominal.

Différents thèmes de recherche peuvent apparaitre via des doctrines scientifiques établies par des institutions françaises spécialisées dans la prévention et la gestion des risques. On peut notamment citer la structuration de la prévention des risques majeurs qui détermine sept piliers de la prévention des risques majeurs [28] :

- Connaissance des aléas et du risque
- Surveillance
- Prévention et éducation
- Prise en compte des risques dans l'aménagement
- Mitigation (atténuer les risques)
- Planification de l'organisation des secours
- Prise en compte du retour d'expérience.

Ou encore le PPRT (Plan de Prévention des Risques Technologiques) [22] publié par le ministère de l'écologie qui met en relief les quatre piliers de la politique de gestion des risques industriels :

- Réduction du risque à la source
- Maîtrise de l'urbanisation
- Organisation des secours
- Information du public.

L'ensemble de ces travaux tend à structurer les problématiques de gestion des risques selon quatre thèmes de recherche :

- Caractérisation et réduction des aléas : Correspond à l'ensemble des actions mises en œuvre permettant une réduction des aléas, ainsi qu'une caractérisation de ces derniers (probabilité d'occurrence, intensité, impacts etc). Cela passe par une caractérisation fine de ces derniers afin de mettre en place des actions pertinentes et adaptées à leurs contexte.
- Vulnérabilité et résistance pour l'évaluation des risques de systèmes sociotechniques : L'objectif de cet axe de recherche est de positionner les enjeux au centre de l'évaluation des risques en s'intéressant à leurs vulnérabilités et leurs résiliences.
- Ingénierie des systèmes complexes face aux risques : Se focalise sur l'aide conceptuelle, méthodologique et technique à apporter à un métier pour mener à bien ses activités exposées à des risques. Cette thématique se focalise sur la détermination de méthodes de modélisation, et de simulation.
- Gestion de crise : Se focalise sur les mécanismes d'apparition et de développement de situations de crises, ainsi qu'à leur gestion.

On peut situer les travaux apportés dans cette thèse sur l'axe de l'ingénierie des systèmes complexes face aux risques. En effet, notre contexte industriel, proche du domaine technique, impose le développement et la maîtrise d'outils de modélisation et de simulation.

L'analyse et la maîtrise des risques (techniques, économiques, sociétaux, et environnementaux) ainsi que leurs impacts sur le sujet de l'étude et son environnement sont cruciaux pour le succès de tout projet de conception, de développement, de pilotage de systèmes complexes. Tout au long du cycle de vie du projet, différents acteurs mènent des activités nécessitant de comprendre, de modéliser, d'analyser afin de justifier et de décider d'actions à engager. Ces actions sont jugées à risques en fonction de leurs possibles conséquences, de la complexité des systèmes visés, des différentes situations susceptibles d'impacter le fonctionnement et la structure de ces systèmes, de la nécessaire collaboration entre ces acteurs, ou encore des moyens mis à leur disposition pour mener à bien leurs tâches.
En réponse à la question Q1, il s'avère donc nécessaire de proposer un outil pour aller vers une assistance des acteurs, intégrant la dimension d'aléas sur le cycle de vie d'un système.
Le cadre de travail est défini par l'Ingénierie Système (IS) qui promeut une démarche holistique et systémique, intégrant l'ensemble des parties prenantes, permettant de couvrir la totalité du cycle de vie des systèmes, mettant en avant l'intérêt d'une démarche basée sur la création et l'usage de modèles (d'où l'appellation de Model Based System Engineering [29]) et dirigée par des

processus reconnus et même standardisés. Elle permet en particulier, de manière itérative, collaborative et selon une approche multipoints de vue :

- D'étudier et de modéliser les systèmes, leurs composants et les interactions qu'ils entretiennent avec d'autres systèmes et l'environnement ;
- D'évaluer des solutions et leurs impacts sur le système : performance, sûreté, résilience, interopérabilité et sécurité … ;
- De vérifier et valider ces solutions ;
- De simuler pour observer le comportement global résultant ;
- De fournir et tracer les données, informations et connaissances nécessaires ensuite pour prendre les décisions correctives et donc atteindre un niveau de risque acceptable pour les parties prenantes ;
- De prendre en compte le contexte d'une action, les pratiques, et les usages des acteurs. Les pratiques étant considérées comme la somme des connaissances, et les usages comme la somme des comportements et règles issus de l'expérience.

Chacune des phases de la méthode MBSE va jouer un rôle important dans la gestion des risques du projet DIAMANTR mis en place par ALSOLENTECH. L'étude, l'évaluation et la vérification des modèles intégrants les risques sont des étapes cruciales pour les ingénieurs d'ALSOLENTECH. Nous allons devoir identifier dans l'état de l'art les capacités de simulation proposées par les outils et méthodes actuels. Le système est soumis à de nombreuses incertitudes, la principale problématique sera d'obtenir un outil capable de modéliser ces systèmes complexes, tout en intégrant les notions modulaires de risques qui leurs sont associées. Cet objectif se décline à la fois sous la forme de la question Q1 pour ses aspects de gestion des risques, et sous la forme de la question Q3 pour les problématiques de modularité des sous-composants. Ces deux problématiques devront être résolues au cours de ce manuscrit par l'étude d'outils et méthodes de modélisation et simulation dont un état de l'art est présenté ci-après.

2.Théorie de la Modélisation et la Simulation

2.1.Modélisation

Une des premières étapes dans un projet de développement ou de conception d'un système est de modéliser ce système. L'objectif de cette modélisation est principalement de bien comprendre son fonctionnement sans ambiguïté, de pouvoir éventuellement l'analyser en déterminant ses points forts et ses points à améliorer, et en favorisant la discussion lors de sa conception. La notion de modèle désigne plusieurs concepts et répond à différentes définitions. Dans la littérature, on trouve différentes définitions de la modélisation. Pour Kaslow et al. dans [30], une modélisation se caractérise par une représentation formelle de la théorie d'une observation empirique. Un modèle peut donc être une représentation logique, mathématique, ou physique conçu pour reproduire un système ou un processus observé. Selon Fishwick et al. dans [31], modéliser peut se résumer à « abstraire

de la réalité une description d'un système dynamique ». Selon Zeigler et al. dans [32], la modélisation est un processus permettant d'organiser les connaissances portant sur un système donné. Il peut donc également être une description abstraite ou concrète d'un phénomène à simuler, ou une technique mise en œuvre pour résoudre une problématique. Enfin, Law et al. dans [33] nous dit qu'un modèle est un ensemble d'approximations et d'hypothèses structurelles sur la façon dont le système fonctionne.

D'un point de vue systémique, nous pouvons décrire la modélisation comme un ensemble d'instructions, règles, équations et contraintes, dans le but de produire un comportement d'entrée et/ou sortie. Ce comportement a pour objectif de produire des données de valeurs proches de la réalité. On peut en conclure que la modélisation est l'abstraction du monde réel grâce à des méthodes et outils normalisés, des formalismes, des langages, dans le but de résoudre une problématique.
Un modèle peut être de différentes natures pour représenter différents systèmes :

- Une modélisation mathématique peut représenter un sujet physique, chimique, météorologique ou informatique (dans certains cas)
- La modélisation 3D peut être utilisée pour représenter un objet réel dans l'espace ou étudier son comportement sous contraintes
- En pédagogie, la modélisation de la discipline consiste en une représentation simplifiée des objectifs d'enseignement

Dans notre contexte et dans la suite de ce manuscrit, nous nous intéresserons à deux formes de modélisations :

- En informatique, un modèle permet de représenter la structure et la dynamique d'éléments logiciels. Cette modélisation peut être réalisée à différents niveaux d'abstractions avec différents outils (UML [34], modèles de flux de données [35], analyse fonctionnelle descendante [36], etc.)
- Dans le domaine des entreprises, la modélisation des processus consiste à utiliser des outils et méthodes dans le but de structurer les activités et processus de l'entreprise. Parmi ces outils, les plus utilisés dans les domaines industriels et académiques sont les Workflow [37] qui permettent la représentation d'une suite d'opération effectuées par une ou plusieurs personnes.

Dans notre contexte industriel et académique, nous avons identifié les standards SADT, XPDL, BluePrint et BPMN (Business Process Modeling and Notation) pour représenter les Workflows (processus métiers, et processus fonctionnels) [38]. Parmis, eux, la norme BPMN offre l'avantage de représenter un processus d'entreprise de façon graphique assez facilement compréhensible par le plus grand nombre d'utilisateurs.
BPMN [39] a été développé par la Business Process Management Initiative (BPMI) qui a été fusionnée en 2005 dans l'Object Management Group (OMG). L'objectif initial du standard était d'avoir un système de notation compréhensible par l'ensemble des utilisateurs potentiels : côté entreprise (par des employés ou des managers), ainsi que côté ingénierie de développement. Ce langage pivot entre les deux domaines permet de réduire les confusions et faciliter les échanges d'idées.

2.2.Formalismes de classification

Dans le domaine de la modélisation et simulation (M&S), on peut résoudre un problème en se basant sur deux types de méthodes :

- Les méthodes analytiques permettent de rechercher une solution générale, mais se heurtent au problème de la complexité du modèle.
- Les méthodes basées sur la simulation permettent de traiter des modèles plus complexes, mais se limitent à des solutions particulières.

Les différents formalismes pour représenter un système dépendent de la façon dont le modélisateur va choisir de gérer l'espace et le temps. Ces formalismes sont choisis en fonction du contexte, des concepts à représenter, de l'étude à réaliser et des objectifs de modélisation qui sont fixés [40].

La classification temporelle détermine la façon dont le temps va être géré dans le modèle. Il peut être :

- À temps discret : le temps progresse avec une mécanique de « pas ». La variable du temps progressera par des valeurs qui seront des multiples du pas. Plus le pas est fin, plus la simulation sera précise ;
- À temps continu : la variable du temps progresse selon des valeurs réelles dictées par le modèle.

La classification spatiale détermine l'espace que peuvent prendre les variables du modèle. Elles peuvent être :

- Des variables discrètes : l'ensemble de leurs valeurs est fini
- Des variables continues : l'ensemble de leurs valeurs est l'ensemble des réels.

En fonction de la combinaison de ces deux classifications (espace et temps), on se retrouve dans différents cas qui vont impliquer des formalismes différents (Table 1).

- Formalisme à équations différentielles : Temps et espace continus. Les équations définissent les changements d'états.
- Formalisme à temps discret : Temps discret, espace continu. Les équations définissent également les changements d'états.
- Formalisme à évènements discrets : Temps continu et espace discret. Les variables ne peuvent changer d'état qu'au moment d'un « pas » temporel. Ce moment est appelé un évènement.
- Formalisme de machines à états finis : temps et espace discret, également nommé machine à états finis et synchrones.

Table 1 : Différents types de modèles

Espace	Discret	Machines à états finis	**Modèles à évènements discrets**
	Continu	Equations aux différences	Equations différentielles
		Discret	Continu
		Temps	

Dans le cadre de ce travail de recherche, nous utiliserons des modèles à évènements discrets.

2.3.Simulation

Une simulation est conçue pour étudier le ou les résultats d'une action sur un élément sans réaliser l'expérience sur l'élément réel. Il s'agit d'un système de calcul numérique (dans la plupart des cas) compatible avec un ou plusieurs des formalismes présentés dans la Table 1, capable d'exécuter un modèle donné, et d'en prédire son comportement.
Le processus standard de création d'une simulation suit les étapes suivantes : d'abord la définition d'un ou plusieurs problèmes, la formulation d'un modèle conceptuel, la conception de l'expérience initiale, la collecte et la préparation des données, puis la vérification et la validation du modèle [41]. Les phases suivantes consisteront à mener les expériences, à analyser et à interpréter les résultats, et enfin, à documenter les résultats [42].

2.3.1.Le temps dans une simulation

La dynamique est un principe particulier dans une simulation car il existe plusieurs dimensions temporelles relatives les unes par rapport aux autres. Il est important de bien les différencier.

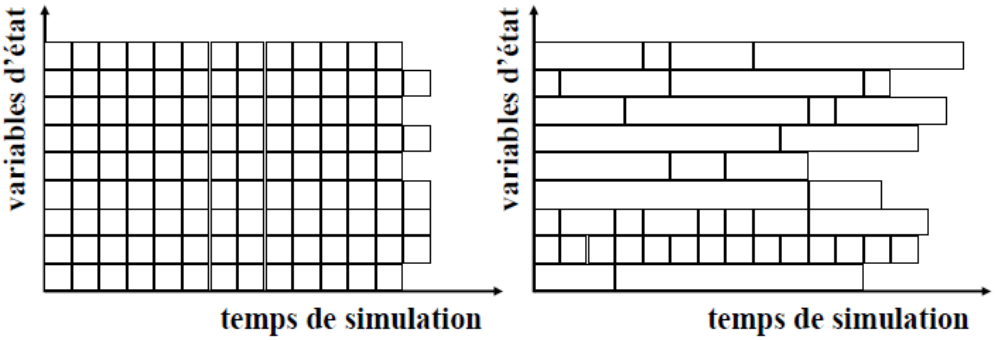
- Le temps simulé est la représentation du temps physique au cours de la simulation. Le temps simulé est défini par le formalisme choisi, et les différents paramètres de la simulation.
- Le temps absolu est le temps réel qui s'écoule pendant l'exécution du modèle par le simulateur. Ce temps est la différence entre la date de début de la simulation, et sa date de fin.

Le temps simulé peut être plus rapide, plus lent, ou s'écouler à la même vitesse que le temps réel. Cette subtilité peut dépendre d'un paramétrage, d'une contrainte physique ou matérielle. Dans le cas d'un modèle complexe à simuler, le temps simulé avancera plus lentement que le temps physique. A l'inverse, un modèle plus simple (ou plus optimisé) pourra faire avancer son temps simulé plus rapidement que le temps réel. Enfin, le temps simulé peut s'écouler à la même vitesse que le temps physique (de façon synchrone ou non), on parle alors de

simulation « temps réel ». Dans ce dernier cas, le temps simulé peut avancer à une vitesse non linéaire. Due à des contraintes logicielles ou matérielles, certaines phases de la simulation peuvent être associées à des modèles simples dont l'exécution pourra être plus rapide que le temps absolu écoulé, et inversement. Dans ce genre de cas, il est donc important de maintenir des points de synchronisation pour que la simulation avance au même rythme que le temps réel.

Nous avons vu plus haut qu'il existe deux méthodes pour gérer le temps dans une simulation :

- Une simulation par temps continu permet de résoudre des problématiques dont la complexité exclue une solution analytique via des équations différentielles.
- Une simulation par temps discret gère le temps de façon échantillonnée. L'ensemble des variables d'état de la simulation ne peuvent changer que sur un des instants du temps simulé. Une simulation discrète peut être dirigée par le temps, ou dirigée par les évènements. Ces deux sous catégories de simulation se différencient par leurs méthodes d'avancement du temps (Figure 1).



Simulation dirigée par le temps Simulation dirigée par les évènements

Figure 1 : Deux façons de gérer le temps

La simulation dirigée par le temps

Dans une simulation dirigée par le temps (ou simulation synchrone), la temporalité est échantillonnée selon un pas défini par l'utilisateur. Le moteur de simulation avancera la variable de temps pas à pas, en exécutant l'ensemble des actions associées à la plage de temps comprise entre les dates t(n-1) et t(n). Chaque pas de temps étant simulé, il est possible qu'aucune action ne soit exécutée à certaines dates. Les actions traitées à une même date sont calculées de façon superposée, ce qui peut entrainer des problèmes de causalité. Il est donc important de bien déterminer la valeur du pas de temps. Un pas trop petit peut entrainer trop de calculs inutiles, un pas trop grand peut amener à ce qu'une action et sa cause soient exécutées au même moment.

La simulation dirigée par les événements

Une alternative à la méthode d'écoulement du temps présentée précédemment consiste à ne pas simuler les dates qui ne sont associées à aucun évènement (simulation asynchrone). On ne simule que les moments ou une action se produit (appelé « événement »). L'écoulement du temps n'est pas échantillonné en pas de temps, chaque action se produit à une date précise stockée dans un calendrier des événements (appelé « échéancier »). Lors de l'exécution de la simulation, le moteur saute de date clé en date clé, sans passer par des étapes intermédiaires. Il est à noter que le calendrier des évènements à venir est recalculé à chaque événement, pour garantir la conformité des lois de causalité. Parmi les différents outils basé sur la simulation dirigée par les événements, on trouve notamment le formalisme DEVS (Discrete Event System Specification) [43] qui décrit un système à événements discrets par des fonctions de transitions d'états.
Une des difficultés de ce mode de simulation réside dans la communication en temps réel. En effet, un évènement provenant de l'environnement extérieur ne pouvant pas être prédit, on ne peut l'inscrire dans l'échéancier de simulation. Il faudra donc dans ce cas, prévoir des points de synchronisation, afin de maintenir deux temps en phase. La simulation dirigée par les événements est un fonctionnement plus adapté à notre besoin de simulation de processus industriels.

2.4.Environnements de simulation

Il existe sur le marché une grande quantité d'outils disponibles. Chacun d'entre eux possède des particularités qui le rendent plus performant dans un certain domaine ou particulièrement efficace pour une problématique donnée. Dans la suite de cette partie, nous détaillons les simulateurs les plus connus du domaine de la modélisation des processus dans le but de les comparer.

2.4.1.Outil et logiciels commerciaux

- FlexSim [44] est un progiciel de modélisation et de simulation à évènements discrets. Il est spécialisé dans la représentation des systèmes industriels tels que les chaines de production, la manutention, la logistique de distribution, le transport etc. L'outil embarque plusieurs bibliothèques d'objets standards, chacun ayant une logique d'exécution préétablie. Son interface graphique permet à un utilisateur de modéliser et de simuler un système sans avoir besoin de coder grâce à son système de palette d'outils. Il embarque un moteur graphique permettant de représenter les simulations en 3D.
- AnyLogic est un progiciel de simulation développé en Java. L'outil fournissant un environnement de programmation graphique, il permet de concevoir des modélisations et des simulations à partir de trois types d'approches : la dynamique des systèmes, les systèmes multi-agents, et les systèmes à événements discrets (les trois approches pouvant être mixées) [45]. Différents types de simulation peuvent être effectués : optimisation, variation de paramètres, comparaison de trajectoires, Monte-Carlo, etc. Enfin, le logiciel donne la possibilité de produire des animations 2D et 3D. En 2002, Borshchev et al. ont présenté un environnement conçu

pour l'outil afin de le rendre compatible avec un standard de simulation distribuée [46].

- Simio est lui aussi un progiciel de modélisation et de simulation basé à la fois sur la modélisation d'objets, et de processus [47]. L'outil fournit une palette de composants prèprogrammé permettant au modélisateur de ne pas avoir à développer ses modèles ou leurs comportements. Capable d'animer un processus en 3D, Simio implémente également de nombreuses fonctions de bruit pour permettre à ses utilisateurs de manipuler le hasard, et donc de jouer avec les imprévus.
- Simul8 [48] est un progiciel de simulation à évènements discrets orienté vers le principe des jumeaux numériques. En effet, l'outil est conçu pour planifier, optimiser, concevoir ou re-concevoir un ou plusieurs systèmes réels de production. Le simulateur permet également, via une palette d'outils, de créer le modèle numérique d'un système en prenant en compte des contraintes réelles comme les taux de défaillance, les capacités maximales, ou n'importe quels autres facteurs extérieurs.

Chacun de ces outils présente des avantages considérables en termes d'interface utilisateur et de fonctionnalités près-implémentées. Mais leurs modèles économiques (licence payante et code source restreint) présentent une limitation pour notre contexte : ils ne sont pas conçus pour être étendus et personnalisables. Il existe également des outils de simulation à licences open source, plus souvent développés et utilisés dans le domaine académique.

2.4.2.Outils et logiciels gratuits et open source

- JaamSim [49] est un outil de modélisation et de simulation basé sur les évènements développés en java. Cet outil est souvent utilisé dans le domaine de la recherche pour sa transparence et son efficacité. Un des avantages de l'outil est son accessibilité car il est open source et permet donc d'être connecté à des outils tiers. Cependant, par défaut JaamSim n'implémente pas d'interface de communication avec l'extérieur (bien qu'il ait fait l'objet de recherches à ce propos pour le rendre compatible avec des standards de simulation distribuée [50]).
- DESMO-J (Discrete-Event Simulation Modelling in Java) est une librairie permettant comme son nom l'indique d'implémenter une simulation à évènements discrets. Elle est développée et soutenue par l'Université de Hambourg [51] depuis 1999 dans le but de fournir un outil rapide et flexible pour construire des modèles de simulation prenant en charge une vision du monde orientée événements, ou processus. La librairie fournit un ensemble complet de classes java permettant notamment d'implémenter facilement des distributions stochastiques, des modèles statistiques afin de laisser l'utilisateur se concentre sur la spécification du comportement du modèle en termes d'événements ou de processus. L'outil offre un potentiel intéressant pour la gestion des risques d'un point de vue statistique, mais ne présente pas d'opportunités en termes d'interopérabilité avec des composants externes.
- PowerDEVS est un logiciel libre développé par l'université Nationale de Rosario (Argentine) [52] en C++. Il permet à la fois de concevoir des

modèles atomiques DEVS, et de les coupler dans des schéma-blocs graphiques pour former des systèmes plus complexes. Les problématiques d'interopérabilité peuvent être résolues en passant par des modules de simulation distribuée compatibles DEVS [53]. Cependant, il ne permet pas la modélisation simple de processus et ne permet pas une prise en compte des risques dans la simulation.

- Papyrus [54] est un projet open source proposant un environnement de développement et d'édition de modèles UML [34] et SysML[55]. Maintenu par le CEA tech, il est conçu à partir de l'IDE (Integrated Development Environment) Eclipse [54], ce qui lui permet d'avoir une architecture modulaire normalisée. Agrémenté d'un moteur d'exécution fUML, il est capable d'exécuter et simuler un modèle UML. L'outil dispose également d'une large communauté d'utilisateur, ce qui le rend facilement personnalisable. Cependant, il n'est pas nativement compatible avec des standards de simulation distribuée, et n'implémente pas de mécanismes de gestion des risques.

L'ensemble de ces outils permettent de proposer des implémentations open source de différents concepts dans le domaine de la simulation (Simulation dirigée par les événements, prise en compte de lois statistiques dans une simulation, modèles DEVS, etc.). Nous avons fait le choix d'utiliser dans notre travail de recherche l'outil Papyrus pour trois principales raisons qui se démarquent des autres outils :

- Sa base Eclipse qui apporte à la fois une modularité de ses composants, et une large communauté d'utilisateurs, tout en restant open source
- Sa simplicité d'utilisation qui permet de modéliser avec des outils graphiques
- Le standard utilisé : UML qui peut être simulé par son moteur Moka, et étendu grâce aux mécaniques de profils.

L'utilisation de Papyrus est la solution la plus adaptée à nos problématiques. Cependant, on peut observer certaines limitations de l'outil par rapport à nos objectifs. En effet, Papyrus n'est actuellement pas en mesure de répondre à nos différentes problématiques de recherche : Le logiciel de simulation ne permet pas une gestion des risques relatifs aux modèles de processus (question Q1). L'outil n'offre qu'une compatibilité partielle avec certains standards de co-simulation et de simulation distribués (question Q2 et Q3). Cependant, modulaire et facilement adaptable, nous allons étendre l'outil pour qu'il prenne en charge les objectifs que nous nous sommes fixés dans ce manuscrit. Les problématiques de gestion des risques de la question Q1 seront traitées en étendant à la fois le module de modélisation et d'exécution de Papyrus. Dans un second temps, nous adapterons l'outil aux problématiques d'interopérabilité à la multi simulation (évoqué par les questions Q2 et Q3). Cela passera par la connexion de Papyrus avec des standards de co-simulation et simulation distribuée.

2.5.Papyrus

Suite à l'analyse de l'état de l'art, il est apparu que Papyrus est l'outil le plus adapté pour nos problématiques de recherches. Nous détaillerons donc ses fonctionalitées dans cette section.

Basé sur Eclipse, comme on peut le voir sur la Figure 2 l'outil comporte par défaut un module de modélisation, ainsi qu'un moteur d'exécution.

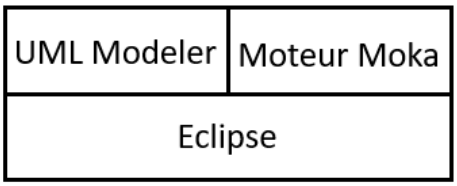


Figure 2 : Architecture Papyrus

Avec son outil d'édition graphique, il permet d'éditer simplement un tous types de diagrammes UML 2.0, intégrant les mécanismes de profils UML et SysML qui permettent de définir des extensions aux standards de base permettant une grande flexibilité de modélisation. Les profils sont des mécanismes propres au standard UML 2.0 permettant d'étendre les métaclasses des métamodèles existants pour les adapter à n'importe quel contexte. Cela inclut donc la possibilité d'adapter le métamodèle UML pour différents domaines (BPMN et SysML étaient à l'origine des profils UML qui ont par la suite été intégrés comme normes par l'OMG).
Papyrus est également accompagné d'un plugin d'exécution de modèles UML : Moka, un moteur d'exécution compatible avec le standard OMG Foundational Subset for executable UML Models (fUML)[56]. L'objectif de ce standard est de servir d'intermédiaire entre les modèles de « surface » (utilisés pour décrire des processus métiers), et les langages « plateformes » (capable d'être exécutés par une machine). FUML est donc un langage pivot nécessaire à la simulation de modèles UML. Dans leur article, Guermazi et al. soulignent quatre préoccupations majeures concernant la norme fUML ainsi que son moteur d'exécution Moka [57] :
:

- Sa connectivité : le moteur et la norme sont difficilement interconnectables avec d'autres outils ;
- Son extensibilité : un profil UML peut nécessiter l'utilisation d'une syntaxe particulière. Cette syntaxe doit être prise en compte par la norme et le moteur ;
- Sa capacité de contrôle : les outils modernes d'exécution et de simulation devraient fournir aux utilisateurs des moyens de contrôler (démarrer, arrêter, suspendre, reprendre, exécuter pas à pas) et d'observer la simulation pendant son exécution. Ces problématiques ne sont pas prises en compte dans la norme fUML (mais sont gérées par le moteur Moka) ;
- Sa prise en charge temporelle : La norme fUML ne se préoccupe pas du temps. [56] souligne : « The execution model is agnostic about the semantics of time. » : fUML ne fait pas d'hypothèses sur les sources d'informations temporelles et les mécanismes qui l'implémentent. Le temps peut être discret ou continu.

Pour compléter et répondre à ces remarques, Ellner et al. soulignent dans [58] les limites de la normes fUML de par son manque de soutien à l'exécution distribuée, et la redondance nécessaire à la complétion des attributs, ce qui peut être source d'erreurs. Dans leur article, les auteurs, proposent une extension fUML capable de prendre en charge l'exécution distribuée.

Papyrus ayant une architecture « en couche », toutes les fonctionnalités implémentées sur les niveaux inférieurs sont utilisables par héritage par les niveaux supérieurs. On peut constater sur la Figure 2 que le moteur Moka et le modeleur UML sont des couches supérieures à l'IDE Eclipse. Ils héritent donc des fonctionnalités implémentées par la base. Cela passe par l'interface graphique modulaire, la structure de débogage (qui permet de fournir à l'utilisateur des fonctions de contrôle, d'observation, et d'animation de l'exécution des modèles), et bien d'autres.
En outre, Moka est lui aussi intégré à la structure de débogage d'Eclipse afin de fournir à l'utilisateur des fonctions de contrôle, d'observation, et d'animation de l'exécution des modèles. Moka peut lui aussi être étendu afin d'exécuter des comportements personnalisés, ou supporter des profils UML customisés par le modélisateur.

Une architecture en couches permet d'implémenter de nouvelles extensions au-dessus d'extensions existantes. Le modeleur UML peut donc être étendu pour supporter les ajouts de nouveaux profils UML. Moka peut lui aussi, être étendu afin d'exécuter des comportements personnalisés, ou supporter des profils UML customisés par le modélisateur.

Développé à partir d'une base solide et très largement utilisée par la communauté, conçu pour permettre une très grande modularité, facilement extensible, simple d'utilisation, Papyrus et sa base Eclipse cumulent les avantages nécessaires à la réalisation de ce travail de recherche par rapport aux simulateurs concurrents.

2.5.1.Architecture logicielle Papyrus & Moka

Papyrus étant construit sur une base Eclipse, il permet une grande flexibilité et modularité de ses composants. Dans ce manuscrit, nous serons amenés à travailler sur une version de Papyrus spécialisée par le CEA Tech, afin de répondre au besoin spécifique d'ALSOLEN TECH.

En effet, le CEA Tech a ajouté une couche logiciel au-dessus de la version standard de Papyrus dans le but de se rapprocher des problématiques métier de l'entreprise, cet outil est baptisé « Diamant » et décrit par la Figure 3.

Les ajouts proposés dans cette thèse sont compatibles avec l'extension Diamant, mais n'utilisent pas les fonctionnalités implémentées par cette couche logicielle. Nous étendons Papyrus à partir de sa dernière version (Diamant), mais les développements produits par le CEA Tech étant « closed source », et dans un objectif de généricité, nos extensions ne s'appuieront pas sur ce niveau. Par

conséquent, toutes les propositions faites dans ce travail de recherche sont compatibles avec la version Papyrus disponible publiquement.

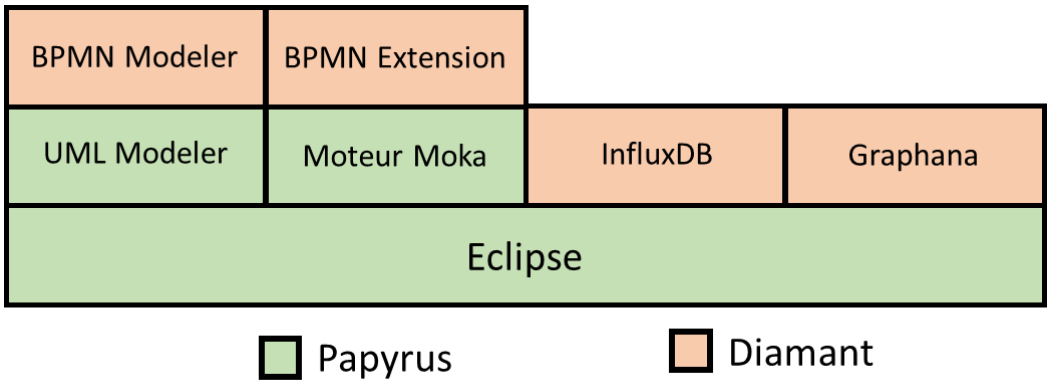


Figure 3 : Couche Diamant de Papyrus

Les différents composants ajoutés par le CEA Tech se structurent en 4 modules :

- « BPMN Modeler » est un profil UML ajouté au-dessus du modeleur UML de Papyrus. Ce profil étend la sémantique native d'UML pour la rapprocher du contexte métier. L'outil de conception de modèle de Papyrus est étendu pour concevoir des modèles proches du BPMN (événements de début, fin, Activités, Sequence flow, Message flow, portes logiques, et commentaires). L'extension ajoute également la notion de temporalité dans une activité BPMN afin de permettre une simulation.
- « BPMN Extension » est une extension du moteur Moka permettant d'interpréter et de simuler les diagrammes UML étendus par le profil UML décrit par la Figure 3.
- « InfluxDB » est une base de données conçue pour les séries chronologiques.
- « Graphana » est un logiciel de visualisation de données qui permet de réaliser et consulter des tableaux de bords à partir d'une ou plusieurs sources de données : InfluxDB.

L'interface de l'outil Diamant se compose de différents éléments identifiés sur la Figure 4 :

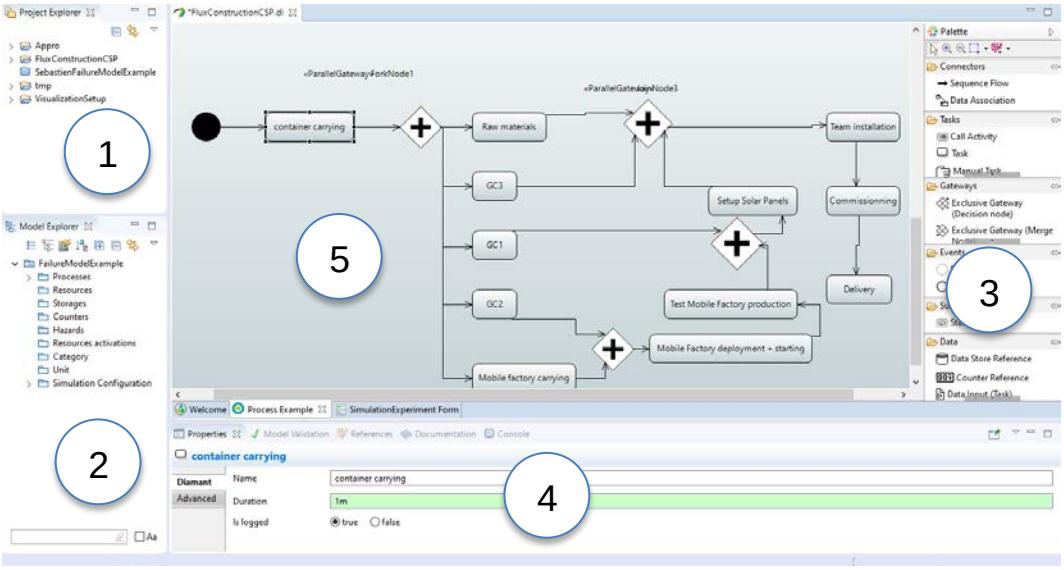


Figure 4 : Capture d'écran outil Diamant

On peut dinstinguer sur la Figure 4 l'interface principale de l'outil se divise en 5 fenêtres.
Le panneau en haut à gauche (1) est un explorateur (ou « Project Explorer ») permettant de naviguer parmi les différents projets chargés dans le répertoire de travail, ainsi que parmi les fichiers qui composent un projet.
Une fois un projet ouvert, la fenêtre « Model Explorer » (2) permet de naviguer dans une arborescence qui le décrit. C'est via cette arborescence que l'utilisateur accède aux modèles, aux ressources, ainsi qu'à la configuration du moteur d'exécution.
La zone de droite est une palette d'outils (3) permettant de faire glisser des composants de modélisation sur la fenêtre centrale (5) de l'outil. C'est avec ces outils que l'utilisateur construit le squelette d'un modèle.
Enfin, la fenêtre du bas (4) permet d'accéder aux propriétés d'un composant sectionné dans la partie centrale (5) de l'outil. C'est via cette fenêtre que chaque tâche qui constitue un modèle va être décrite en termes de ressources, temporalité, etc…

Comme nous l'avons dit plus haut, Papyrus est un outil travaillant de pair avec le plugin Moka, son moteur de simulation. Afin de comprendre l'articulation entre ces deux composants, il est important de clarifier le fonctionnement des « *Advice* ».

L'éditeur graphique de Papyrus est compatible avec le standard UML 2. Lors de l'exécution d'un modèle conçu avec l'éditeur graphique de Papyrus, le diagramme UML est envoyé au module d'exécution Moka, lui-même compatible UML et fUML (Foundational UML) [56]. FUML est un sous ensemble du langage UML pour lequel il existe une sémantique d'exécution standardisée par l'Object Management Group (OMG). Un modèle construit en fUML est donc exécutable comme l'est un programme écrit dans un langage plus traditionnel. Il existe également une syntaxe textuelle standardisée pour fUML appelée « Action Language for fUML » (Alf) [59], particulièrement utilisée pour définir des comportements détaillés dans le contexte d'un modèle fUML et procéder à la simulation.

C'est via ce langage pivot que le moteur d'exécution Moka va être en mesure d'exécuter le diagramme construit par l'utilisateur.
Ainsi, la Figure 5 illustre une succession de transformations de modèles opérés par différents acteurs, tous connectés les uns aux autres.

Figure 5 : Transformation de modèle depuis BPMN vers une simulation exécutée

Le langage inspiré du BPMN implémenté par le CEATech (sous Diamant) est un profil du langage UML, converti en fUML puis Alf par le moteur d'exécution Moka. Ce code est alors exécuté, puis les résultats sont transmis à la base de données InfluxDB.

2.6.Outils de simulation du risque

Les outils de gestion du risque sont pour la plupart issu des méthodes formelles, tels : Obeng, risk checklist, brainstorming [58], ISSRM [57], et les diagrammes d'influences [62]. Ils sont fréquemment implémentés dans des serious games afin de sensibiliser leurs utilisateurs aux problématiques qu'ils soulèvent.

Dans leur article, Pontakorn et al. [63] présentent un serious game qui aide à simuler le management du risque dans le domaine du génie logiciel. Durant la simulation, les participants suivent les principes de la gestion des risques et visent à atteindre leurs objectifs tout en économisant leurs dépenses.

Navarro et al. présentent dans [64] un simulateur serious game baptisé SimSE destiné à l'enseignement universitaire. L'objectif de SimSE est de combler le fossé entre la grande quantité de connaissances conceptuelles sur les processus logiciels données aux étudiants en cours, et la quantité comparativement faible de ces connaissances, qu'ils mettent en pratique dans le cadre d'un projet de génie logiciel. L'outil accomplit cela en permettant aux étudiants de pratiquer, par le biais d'un simulateur, l'activité de gestion de différents types de processus de génie logiciel.

En raison de l'utilisation croissante des méthodes agiles, l'enseignement de SCRUM en tant que méthodologie de gestion de projet est devenu de plus en plus important. C'est la raison pour laquelle Gresse Von Wangenheim et al. présentent dans [65] un serious game destiné aux universités pour enseigner la méthode SCRUM baptisé SCRUMINA.

Dans la continuité de ces formalismes et du point de vue d'une analyse fonctionnelle, certaines approches de la modélisation prennent aussi en compte le risque en exploitant un langage non dédié pour les représenter.

Cependant, Jurjens et al. observent dans [66] que les considérations en matière de sécurité se posent souvent après la définition des processus métiers. C'est en partant de ce constat que Olga Altuhhova et al., soulignent dans [67] à quel point les préoccupations de sécurité sont négligées dans les processus de modélisation d'entreprise, notamment lors des modélisations de systèmes d'information. Leur objectif est d'utiliser le standard BPMN pour modéliser la sécurité des processus commerciaux et suggérer des extensions au langage pour l'orienter vers des capacités de gestion des risques de sécurité. Pour ce faire, les auteurs ont sélectionné le modèle de domaine IS Security Risk Management (ISSRM) et ont décidé de l'aligner sémantiquement sur le standard BPMN en passant par le développement d'une extension dédiée.

Un modèle de domaine est un système d'abstractions qui décrit certains aspects d'un domaine de connaissance. Dans le cas d'ISSRM [68], [69] (représenté par la Figure 6) c'est un modèle conçu pour avoir une approche rigoureuse de la gestion des risques dans les systèmes d'information.

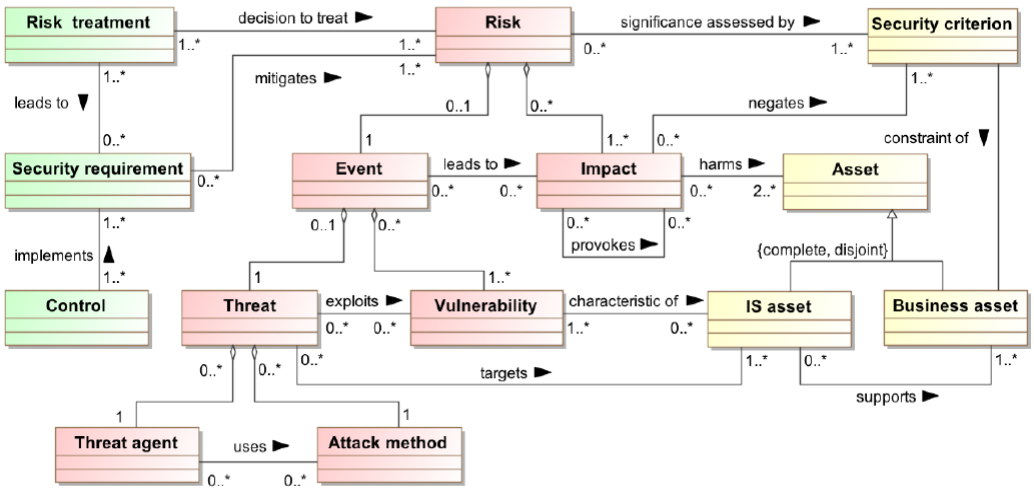


Figure 6 : Modèle de domaine ISSRM

Son fonctionnement est divisé en trois parties :

- Identification des atouts (en jaune) : Ces classes servent à identifier (et décrire) les points forts de l'entreprise et leurs critères de sécurité. On entend par « atout » (ou Asset) tout ce qui a de la valeur et qui permet à l'entreprise d'atteindre son objectif. Pour chacun des points importants, on va lister les informations, les processus, et les capacités essentielles nécessaires à l'entreprise pour les atteindre (Business asset), ainsi que les modules / fonctionnalités du système d'informations permettant de les supporter (IS asset). Enfin, pour chaque Business assets, on va répertorier les critères de sécurité que l'entreprise désire (comme la confidentialité, l'intégrité des données, la disponibilité, etc…).
- Identification des risques (en rouge) : Ce groupe de classes identifie l'ensemble des risques liés aux critères de sécurité. Chacun de ces risques peut être déclenché par un évènement qui aura un ou plusieurs impacts sur les atouts du système. Ce modèle nous propose de pouvoir ajouter la description de menaces pour chaque évènement ainsi qu'une « méthode d'attaque » : un moyen par lequel une menace est exécutée.

- Traitement du risque (en vert) : Ces classes identifient et déterminent quelles sont les actions à mettre en place pour traiter le risque. Il s'agit d'une décision à prendre (l'évitement, la réduction, le transfert) pour annuler le risque ciblé. Chacun de ces traitements amène à des exigences de sécurité qui permettent de maintenir des niveaux de dangers maitrisés.

Les auteurs de [67] considèrent que les constructions linguistiques du standard BPMN peuvent être alignées sur le modèle ISSRM. Ils proposent alors une version étendue du BPMN afin de correspondre aux concepts énoncés par ISSRM en termes d'atouts, de risques, de traitements des risques, etc. Cette proposition est accompagnée d'un ensemble d'exemples de cas d'utilisation de l'extension.

Marcinkowski et al., proposent également dans [70], une extension du langage BPMN visant à intégrer et modéliser les risques dans les modèles d'entreprise. Dans l'ouvrage, les auteurs divisent les risques en trois catégories :

- Les risques d'entreprise, de nature stratégique, visant à protéger une activité afin d'assurer la continuité de ses opérations commerciales, ainsi que le respect des réglementations industrielles et gouvernementales ;
- Les risques liés aux données, qui portent sur la disponibilité des informations et leurs sémantiques, quels que soit leurs supports ;
- Les risques liés aux événements de tout type mettant en danger la pérennité de l'entreprise et de ses activités.

Le standard BPMN se distingue des autres langages du fait de son large éventail d'événements différents. En effet, au-delà des événements de base utilisés dans la majorité des cas (événement de début et de fin), il propose douze types d'événements différents. Parmi cet ensemble, c'est l'événement d'erreur (visible sur la Figure 7) qui est utilisé dans l'article [70]. Marcinkowski et al. nous précisent que la fonctionnalité de l'évènement « erreur » peut être utilisée dans le but d'attribuer des risques identifiés aux processus, sous-processus, ou activités décrits par le langage.

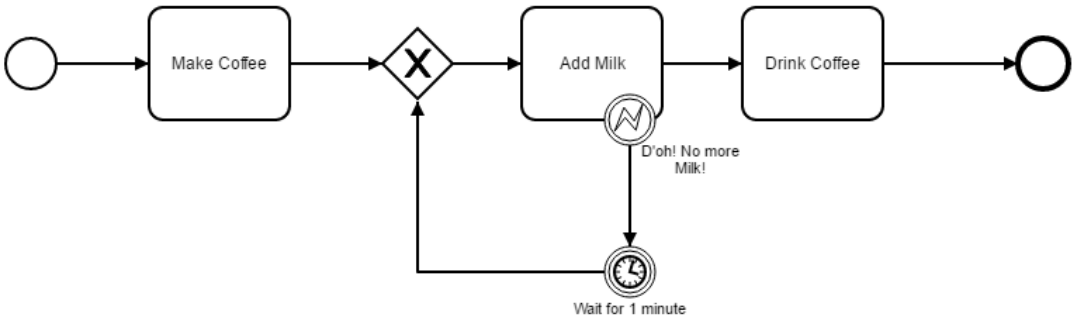


Figure 7 : Symbole « Erreur » dans le standard BPMN

L'indication d'une erreur est donc le point de départ de l'extension du standard que proposent les auteurs. La norme BPMN n'étant pas formelle sur l'identification des risques, ils seront annotés et soumis à un métamodèle proposé par les auteurs (voir classes grises du métamodèle Figure 8) appelé Facteur de risques (ou « RiskFactor »). Du point de vue du métamodèle BPMN, le facteur de risque est considéré comme l'enfant d'un « Artefact » BPMN et sera associé à un « sequenceFlow ». De la même manière que dans leurs publications connexes

[71], [72], la classe « RiskFactor » dispose de deux attributs : « occurenceProability », et « impact » qui sont représentés par deux entiers variant de 1 (risque faible) à 5 (risque élevé) pour représenter les niveaux de dangerosité et d'occurrence de chacun des risques désignés.
La Figure 8 présente le métamodèle des facteurs de risques BPMN. On remarque sur la figure qu'un facteur de risque est l'élément enfant d'un artéfact BPMN et sera complété par des propriétés supplémentaires.

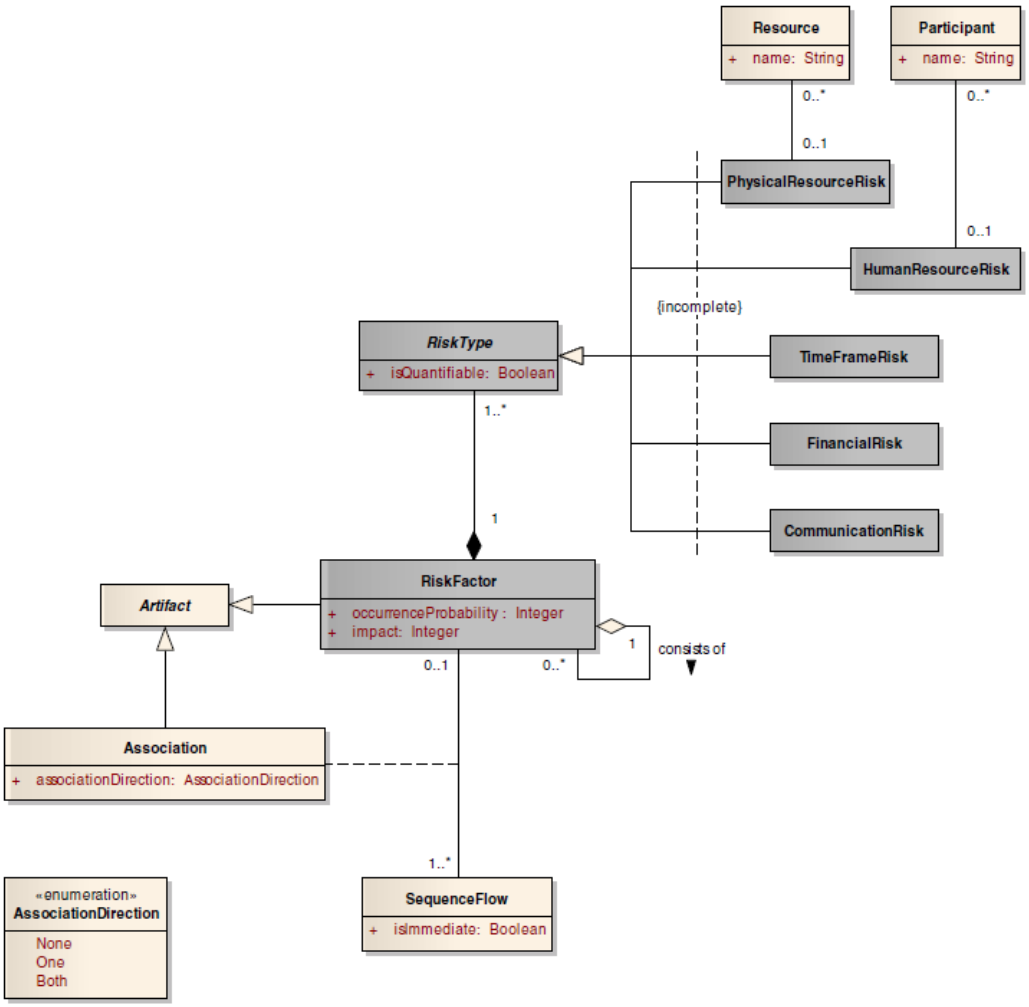


Figure 8 : Métamodèle des facteurs de risques BPMN

Nous venons de citer les deux paramètres « occurenceProbability » et « impact ». Un facteur de risque est également composé d'un ou plusieurs types de risques (ou « RiskType ») qui peuvent prendre plusieurs formes : la norme suggère 5 types de risques, non exhaustifs, et entendables en fonction du contexte.

- Risque lié aux ressources physiques ;
- Aux ressources humaines ;
- Au temps ;
- Au budget ;
- A la communication.

Dans cette extension BPMN, les auteurs proposent de traiter chaque risque identifié. Pour ce faire, un autre type de tâche est ajouté au métamodèle : « RiskHandler » (voir Figure 9).

Étant donné qu'une tâche hérite des caractéristiques d'une activité, on peut constater sur la Figure 9 que RiskHandler peut accéder aux ressources associées à l'activité incriminée. De plus, le paramètre « *mitigationMethod* » référence une énumération qui contient une liste de différentes méthodes à appliquer pour gérer le risque désigné.

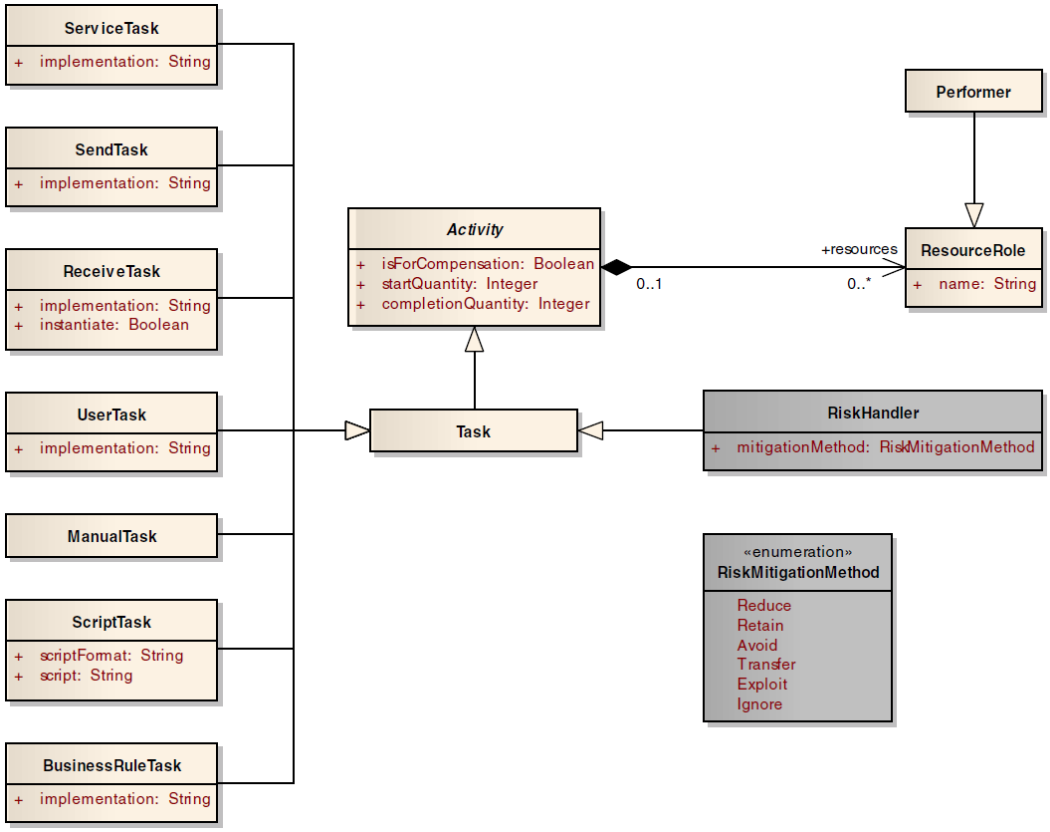


Figure 9 : Métamodèle RiskHandler BPMN

La définition d'une extension BPMN permettant la prise en charge de la modélisation du risque est une solution ayant l'avantage de prendre en compte les risques directement lors de la phase de modélisation des processus d'entreprise. La proposition de métamodèle faite par les auteurs n'est pas en désaccord avec la norme BPMN maintenue par l'Object Management Group OMG et permet donc de modéliser les risques dans le processus d'une entreprise, tout en étant en accord avec le standard. Cependant, BPMN étant un langage de haut niveau, il ne permet pas d'être exécuté.

Comme nous l'avons évoqué précédemment, la modélisation d'un système est généralement suivie d'une phase de simulation. Dans le cas où le standard choisi est un langage exécutable, ces deux étapes sont une suite logique. Dans d'autres cas, on effectue une première modélisation au niveau métier dans un standard ne permettant pas la simulation (comme le BPMN), pour ensuite avoir recours à une seconde modélisation, ou une transformation de modèle pour obtenir une représentation du système exécutable. Nous venons de voir qu'il existait différents moyens d'inclure la gestion des risques dans une modélisation BPMN, il est également pertinent de s'intéresser à la simulation de ce dernier.

Ralf Laue et al. présentent dans [73] une analyse et une critique de la spécification BPSIM. Pour les besoins de la simulation, un modèle doit contenir un certain nombre d'informations (temporalité, gestion des ressources, événement de début, de fin, etc.) [74]. Le standard BPSIM [75] est une extension du langage BPMN permettant la simulation de processus. Pour ce faire, des nœuds BPSIM sont ajoutés au XML du langage de base afin d'ajouter des informations complémentaires et permettre une simulation du langage. Ralf Laue et al. stipulent que le langage permet de modéliser indépendamment des outils de simulation, cependant, lors des tests pratiques, ils constatent que certains outils qui proposent une implémentation du standard ne mettent en œuvre qu'un sous-ensemble de la norme. D'autre part, certaines implémentations fournissent des extensions utiles mais propriétaires. De plus, la spécification étant relativement récente, des problèmes d'interface utilisateurs apparaissent (comme la nécessité de devoir passer par l'édition de fichiers textes pour simuler certaines ressources).
BPMN et BPSIM sont des standards permettant de modéliser des systèmes et des entreprises dans une dimension statique. BPSIM permet de simuler des variables aléatoires, mais les distributions de ces variables ne peuvent pas être dynamiques durant une simulation. Les auteurs nous précisent également que BPSIM ne permet pas de décrire précisément des ressources comme peuvent le faire des outils de simulation traditionnels.

On trouve également dans la littérature des outils et des simulations conçus pour répondre à des problématiques spécifiques reliées à des domaines précis.

Dans [76], Ming et al. s'intéressent à l'évaluation des risques écologiques selon une approche probabiliste pour caractériser le risque et traiter l'incertitude en utilisant des simulations de Monte Carlo. Cette simulation s'appuie sur des données de l'armée américaine pour évaluer les distributions probabilistes des paramètres de risques liés à l'exposition d'uranium appauvri. Basé sur un outil développé par Haiyi Lu [77], l'outil est conçu pour étudier des modèles représentant des éco-risques. Soucieux des problématiques d'interface utilisateur, il est conçu en Visual Basic 6 pour permettre une interface graphique calquée sur Windows et gère ses données avec Microsoft SQL Server. On remarque que l'ensemble de l'outil, son architecture UML, ses bases de données sont conçues pour traiter des risques écologiques.

Dans [78], Bixio et al. présentent le cas des stations d'épuration des eaux usées Belges qui, dans une démarche de rénovation d'une structure, font face à de nombreux aléas. L'article décrit une méthode qui allie une simulation probabiliste (de Monte Carlo) couplée à des simulations déterministes (voir Figure 10).

1- Dans un premier temps, des lois de distribution de probabilités sont réparties sur l'ensemble des variables d'entrée du système afin de représenter l'ensemble des cas possibles.
2- Pour chaque entrée, un moteur de simulation Monte Carlo attribue une valeur selon les courbes de distributions générées précédemment. L'ensemble de ces valeurs est donné en entrée du modèle déterministe.

3- Le modèle déterministe est résolu pour chacun des cas soumis par le moteur stochastique.
4- Les résultats de chacun des modèles sont stockés, et le processus est répété suffisamment de fois pour observer l'apparition de lois statistiques.

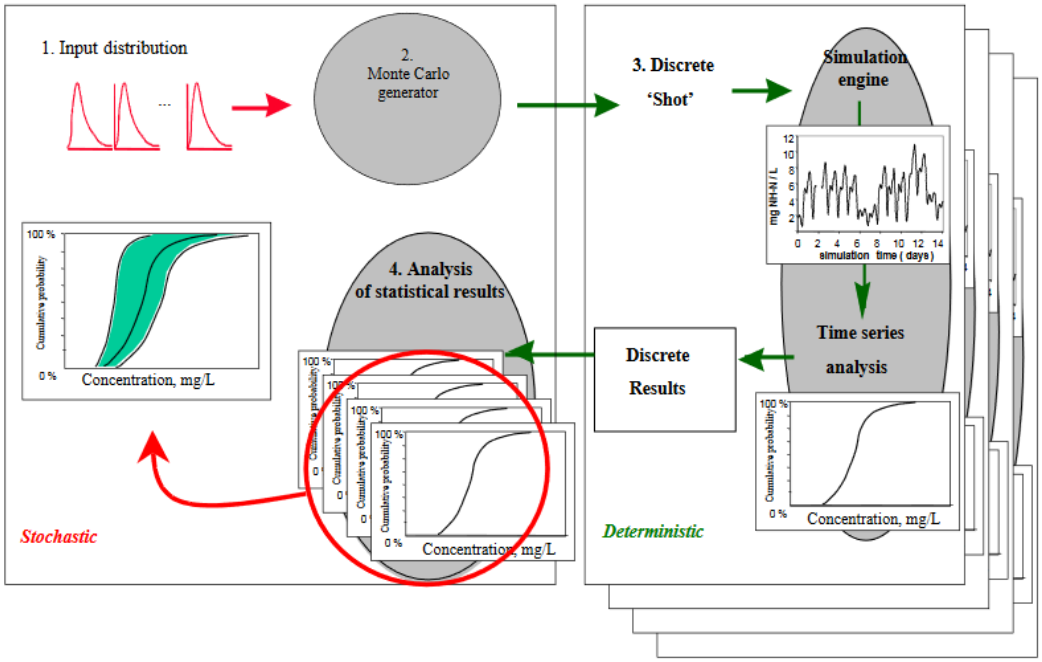


Figure 10 : Cycles de simulations hybrides stochastiques et déterministes

Enfin, on peut trouver dans la littérature des outils plus génériques destinés à la modélisation et à la simulation du risque.

Originaire des prototypes IO-SUIT [79], et Pro-(R)isk [80], l'outil RIO-Suit hérite d'une architecture modulaire (que l'on peut voir sur la Figure 11). Commune à plusieurs projets de recherche, l'outil propose de supporter la gestion des Risques et l'Interopérabilité au sein de collaborations d'Organisations (RIO).

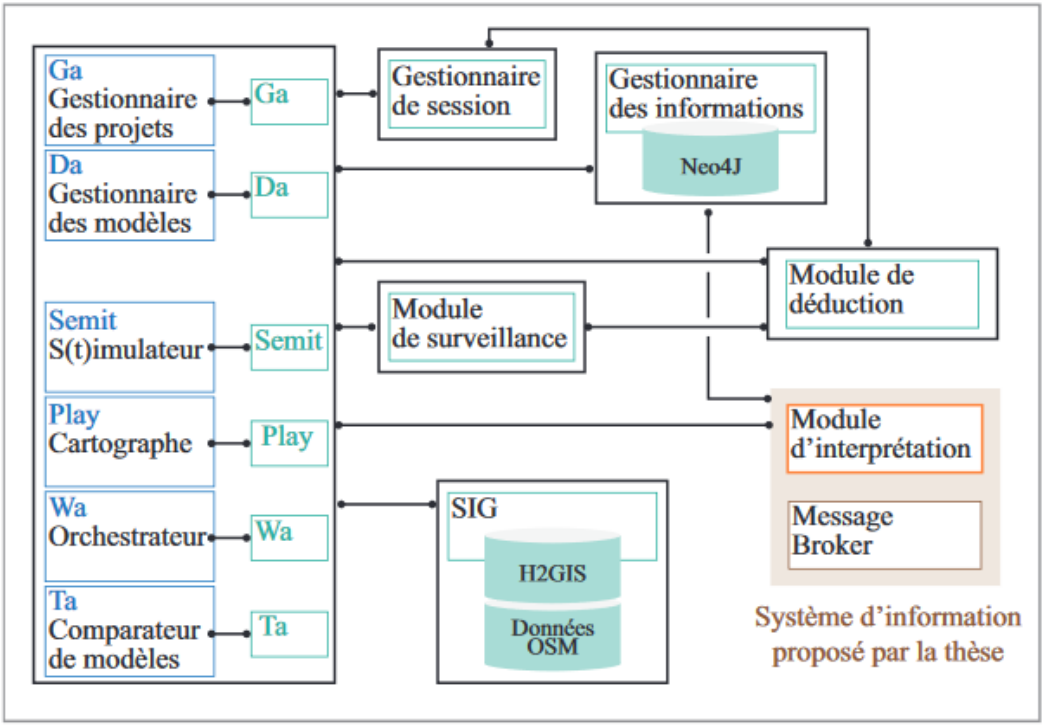


Figure 11 : Architecture RIO-Suit

Via un module de situation, l'utilisateur peut déclarer le modèle d'un système collaboratif sous forme de graphes (Neo4J) qui incluent un ensemble d'objectifs lié à cette collaboration. A partir de ce modèle de situation, le module de déduction propose un processus adapté aux objectifs de collaboration [81]. L'orchestration de la simulation est gérée par un module « workfows assistant » qui permet au module de surveillance un suivi des données de l'utilisateur (RIO-Ta) en comparaison constante avec un modèle de situation de crise. Doté d'un Système d'Information Géographique (SIG), RIO-Suite permet de déduire en fonction d'un contexte collaboratif un ensemble de risques et leurs probabilités.

Tsiga et al. proposent dans [82] un outil de simulation d'aide au management du risque dans un projet. L'outil permet aux participants de mieux comprendre un projet et d'analyser les risques en utilisant différentes méthodes telles que l'exposition au risque, et la méthode de Monte Carlo. A partir des résultats de l'étude, l'outil aide à améliorer la compréhension et la mise en œuvre de la gestion des risques dans les projets.

L'outil développé est conçu pour fonctionner sur un client léger (voir Figure 12). L'utilisation de ce dernier est prévue pour fonctionner en 3 étapes. Dans un premier temps, l'outil interroge l'utilisateur sur la nature de son projet via un système de questionnaire. Ces questions sont utilisées pour recueillir des informations génériques sur le projet, notamment sur la classification « Obeng »[80] [81], ainsi que sur le budget, le management d'équipes, les risques connus, les risques inconnus, etc. Une fois l'ensemble du questionnaire achevé, un ensemble de suggestions basées sur les recommandations Obeng est donné.

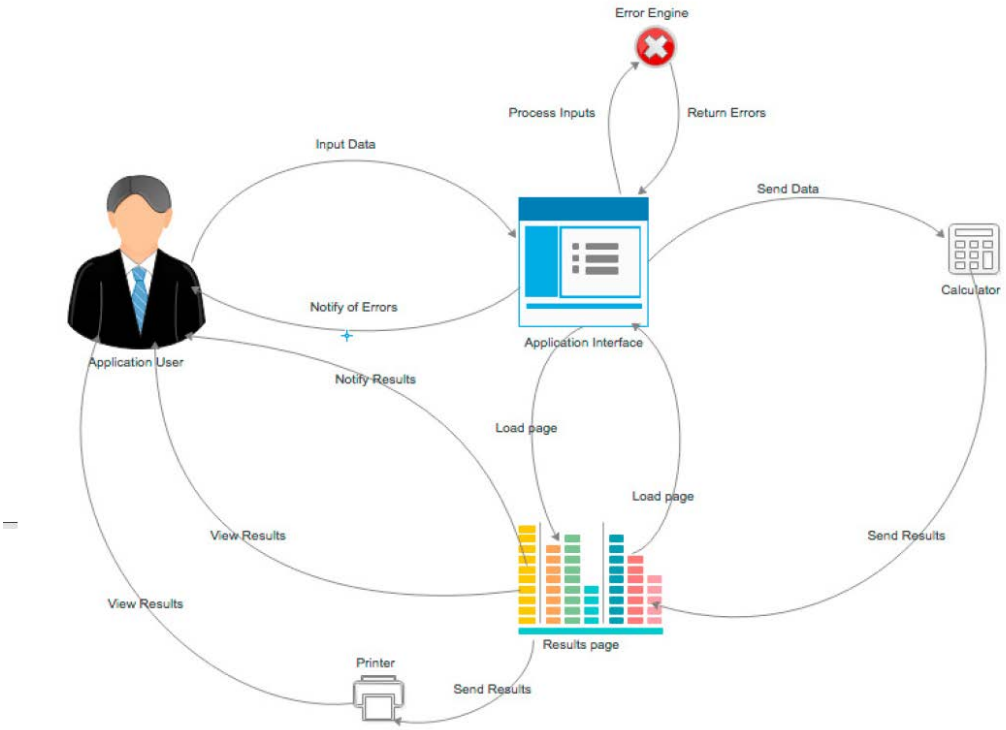


Figure 12 : Diagramme de contexte de l'outil du point de vue de l'utilisateur

Dans un second temps, l'utilisateur accède à une interface de customisation des risques individuels. Il peut alors ajouter des informations pertinentes sur chacun d'entre eux afin de mieux détailler leurs impacts sur le projet. L'outil calcule récursivement la dangerosité de chacun d'entre eux, et les trie par ordre décroissant.
La dernière étape d'utilisation de l'outil consiste à visualiser le tableau récapitulatif des risques. Depuis ce tableau, l'utilisateur peut ajouter des options d'atténuation des risques, des contraintes calendaires, des probabilités d'occurrence, etc. Il peut également sélectionner les risques qu'il veut visualiser dans l'étape suivante : la simulation.
Une fois tous les indicateurs configurés, l'utilisateur peut choisir d'effectuer deux simulations Monte Carlo : Une simulation récursive de 1000 itérations basée sur les calculs de coûts et durée, l'autre qui prend en compte le coût numérique des risques encourus ainsi que le coût de base du projet, qui offre une vision des risques plus globale.

Better et al., présentent dans [85] une méthode qui associe l'utilisation des principes d'optimisation mathématique et d'optimisation de simulation afin d'aider les gestionnaires de risque à atteindre leurs objectifs. Dans ce manuscrit, l'auteur part du principe que dans un scénario à risque, les différents paramètres à prendre en compte ne sont pas toujours connus et peuvent varier dans une mesure plus ou moins grande, selon la nature des phénomènes qu'ils représentent. A partir de là, les approches traditionnelles d'optimisation telles que l'optimisation par scénario, ou l'optimisation robuste sont efficaces pour trouver une solution réalisable pour tous les scénarios envisagés. Cependant, ces méthodes ne sont fonctionnelles qu'avec un petit nombre de scénarios possibles. La taille et la complexité des modèles qu'ils peuvent traiter sont limités. Dans ces cas,

l'optimisation de la simulation devient la méthode de choix. L'optimisation de la simulation permet d'imposer des exigences sur les résultats de simulation, une fonctionnalité que les méthodes basées sur les scénarios ne peuvent pas proposer. Cette démarche propose la séparation de la méthode du modèle. Le problème d'optimisation est défini et traité en dehors du système complexe. Par conséquent, l'évaluateur (c'est-à-dire le modèle de simulation) change et évolue pour intégrer des éléments supplémentaires du système, tandis que les routines d'optimisation restent les mêmes. Il y a donc une séparation complète entre le modèle qui représente le système, et la procédure utilisée pour résoudre les problèmes d'optimisation définis dans ce modèle.

On observe que dans un contexte à haut risque, la prise en compte des incertitudes est à faire dès les premières phases de modélisation. Plus elles sont faites en amont, mieux l'entreprise sera en mesure d'anticiper et d'éviter ces événements non désirés. Après une analyse de l'état de l'art, la prise en compte des incertitudes implique deux difficultés :

- Ajouter une représentation des risques et les paramètres qu'ils impliquent (impacts, occurrences, probabilités, effets secondaires) à la modélisation d'un système augmente drastiquement la difficulté de modélisation. Certains auteurs proposent l'externalisation de la génération des aléas et des paramètres d'entrée d'une simulation afin de délocaliser cette complexité illustrée par la Figure 13. Cependant, le modèle n'est plus autonome et ne peut être exploité sans son scénario de risques.

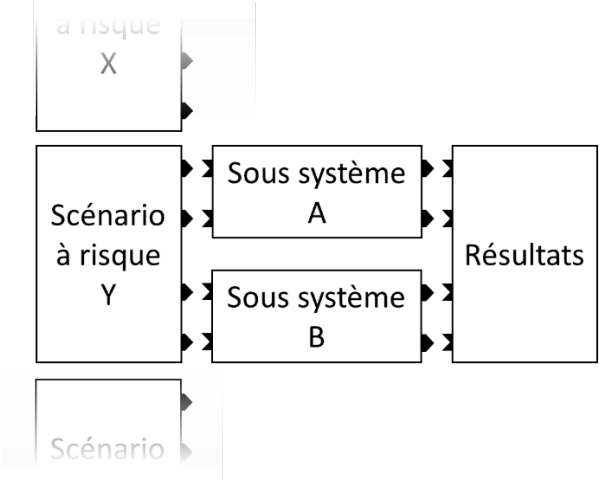


Figure 13 : Gestion des paramètres à risques d'une simulation

- Simuler un système global implique l'accès et la communication avec la totalité des sous-systèmes qui composent le sujet. Dans une très large majorité des cas, les différents sous-systèmes proviennent de domaines vastes et variés. Il y a donc une problématique d'homogénéité des composants non négligeable.

Ces deux difficultés font référence à deux des questions de recherches évoquées dans ce manuscrit :

- La problématique de complexité de représentation des risques au sein d'une modélisation systémique est décrite par la question Q1. Nous étudierons plus tard dans ce manuscrit un moyen de séparer les données

d'entrée, et la gestion des risques qui les concernent (illustré par la Figure 14). Nous inclurons à la description d'un sous-système, ses propres données d'entrées dans un état de fonctionnement nominal permettant d'obtenir un modèle autonome et facilement exportable. La gestion des risques et des aléas relatifs au système global (et les sous-systèmes qui le composent) pourra ainsi être décrites dans une base de données des risques et aléas configurable par l'utilisateur. Cette base de données pourra être connectée, ou non, au modèle nominal et peut évoluer au cours de la simulation.

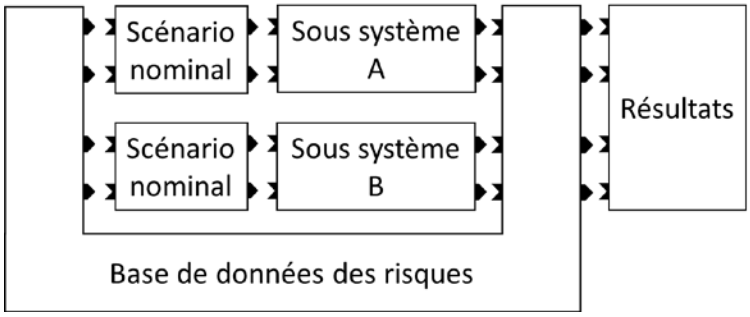


Figure 14 : Séparation des données nominales et des risques d'une simulation

- La problématique induite par l'hétérogénéité des composants à simuler sera étudiée par les questions Q2 et Q3 de ce manuscrit. Nous étudierons en suivant les outils et méthodes conçus pour faire faces aux enjeux d'interopérabilité.

2.7.Interopérabilité

Un système industriel étant la combinaison de différents sous-systèmes, nous nous intéressons aux problématiques d'interopérabilité et comment les dépasser. La littérature propose différentes définitions de l'interopérabilité. Selon l'IEEE [86], la notion d'interopérabilité peut se définir par la capacité que peuvent avoir plusieurs systèmes ou composants à échanger des informations. Deux systèmes interopérables peuvent, actuellement ou à l'avenir, parfaitement se comprendre et travailler entre eux sans aucune restriction. Le terme est initialement défini pour parler de technologies, il s'est plus tard décliné aux facteurs sociaux et plus généralement organisationnels. L'interopérabilité est considérée comme significative si les interactions peuvent avoir lieu à plusieurs niveaux d'abstractions pour un problème donné, dans un contexte professionnel défini et avec une sémantique particulière. Résoudre des problèmes d'interopérabilité consiste à identifier des barrières (obstacles de compatibilité) entre deux systèmes donnés et tenter de les résoudre.

On peut catégoriser les différents concepts de base relatifs à l'interopérabilité en entreprise selon trois axes [87] représentés sur la Figure 15 :

- Les barrières d'interopérabilité
- Les niveaux de préoccupation
- Les approches

Chacun de ces axes peut se décliner selon différentes strates ou échelons.

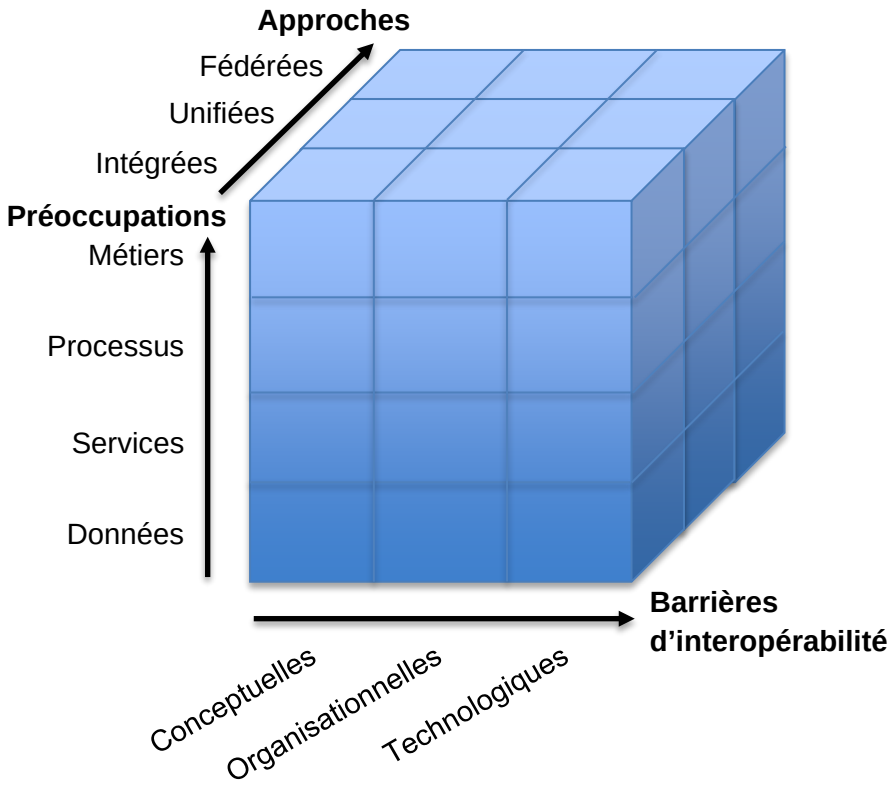


Figure 15 : Axes d'interopérabilité des entreprises

L'axe des barrières de l'interopérabilité se compose de trois échelons [88] :

- Les barrières conceptuelles représentent des problématiques en lien avec la syntaxe et la sémantique utilisées lors des échanges.
- Les barrières organisationnelles représentent les droits et responsabilités accordées à chacun pour que l'interopérabilité puisse avoir lieu.
- Les barrières technologiques représentent tout ce qui est lié aux problématiques de l'utilisation des technologies de communication. On trouve dans cet échelon les normes à utiliser en termes de communication, stockage, traitement des données de façon informatisée.

L'axe des niveaux de préoccupations permet de situer les différents niveaux décisionnels et opérationnels dans lesquels l'interopérabilité doit agir. Cette catégorisation est un héritage de l'architecture ATHENA conçu par Berre et al. [89] pour les systèmes d'informations. Ce principe constitué de quatre échelons peut se généraliser à tout système :

- La préoccupation au niveau « données » de la Figure 15 fait référence à l'implémentation des modèles, ainsi qu'aux langages et requêtes manipulés par le système. A ce niveau, il est important d'avoir un ensemble de dispositifs physiques et numériques compatibles avant de permettre l'échange de données.
- La préoccupation au niveau « services » se focalise sur la composition et le fonctionnement des différentes applications entre elles. Cette partie se focalise sur l'interconnexion des différents modules, et s'assure de leurs compatibilités syntaxiques et sémantiques.
- La préoccupation au niveau « processus » vise à gérer la collaboration de ces derniers (on entend par processus un ensemble de services et

fonctions séquencés pour assurer un besoin spécifique du système). Dans le cas d'une entreprise, ou d'un système complexe, les processus internes doivent parfois être interconnectés et connectés à leurs environnements.

- Les préoccupations « métiers » se chargent de l'organisation des décisions à haut niveau. C'est à ce niveau que sont prises les décisions stratégiques en fonction des contraintes, processus, objectifs et de l'environnement du système.

L'axe des différentes approches regroupe trois méthodes pour aborder les problématiques d'interopérabilité.

- L'approche intégrée. Le système définit un format commun pour tous les modèles et composants amenés à communiquer. Ce format doit être précis, détaillé et accepté par tous les acteurs du système.
- L'approche unifiée. Le système est libre d'utiliser un format de communication qu'il désire, mais il doit spécifier des interfaces / normes pivots avec l'extérieur standardisé. Par exemple, dans le domaine de la simulation distribuée, cela reviendrait à employer le standard HLA.
- L'approche fédérée. Les sous-systèmes ne possèdent pas de formalismes communs, ils doivent arriver à communiquer à la volée. Aucun modèle ou standard n'est imposé à l'ensemble des modules.

De nos jours, que ce soit au niveau métier, ou au niveau technique, la majorité des systèmes tendent à utiliser des approches intégrées ou unifiées en passant par des formalismes le permettant comme Pop* [90], Unified Enterprise Modeling Language [91], [92], ou High Level Architecture [93]. L'ensemble de ces technologies permettent de prendre en charge les problématiques d'interopérabilité horizontales sur la Figure 15.
Il existe pour chacun des niveaux de préoccupation de la Figure 15 des outils et langages permettant de modéliser les questions d'interopérabilités et leurs complexités. La représentation par des paradigmes de modélisation d'entreprise s'approchant du niveau métier peut se faire avec l'aide d'outils comme UEML([94]), GRAI ([95]), ou BPMN, tandis que des outils comme UML, ou workflow se rapprochent plus des niveaux de préoccupations techniques. L'ensemble de ces formalismes a permis de standardiser la spécification des systèmes et des entreprises afin de faciliter le développement de ces derniers.

Il reste cependant la nécessité de transformer ces modèles pour naviguer d'un point à l'autre dans l'espace défini par les repères de la Figure 15. La navigation dans l'espace formé par ces axes se fait avec des méthodes de transformation de modèles et permettent ainsi de traiter les problématiques d'interopérabilité verticales de la Figure 15. Il existe différentes méthodes qui varient en fonction du déplacement que l'on souhaite effectuer.
Model-Driven Architecture (MDA) propose une démarche guidée, orientée conception logicielle, et permet de faire la translation d'une préoccupation de services vers une préoccupation de données (voir Figure 15). D'autres approches sont développées pour d'autres déplacements dans le repère, comme Model-Driven Engineering (MDE) qui réfère à des niveaux plus abstraits sur les axes de préoccupations et de barrières d'interopérabilités [96]. Dans notre cas, nous avons utilisé la méthode Model Based System Engineenring (MBSE) qui se focalise sur

la création et l'exploitation des modèles de domaines pour communiquer entre des couches de haut niveau (métier, processus), et des couches de plus bas niveau (services, données). Ces méthodes nous permettront de convertir un besoin (défini avec ALSOLENTECH) pour concevoir des modèles formels dans l'outil de modélisation et de simulation.

Comme nous l'avons vu dans cette partie de l'état de l'art, de nombreux outils de simulation existent, mais aucun ne prend en compte à la fois les problématiques d'interopérabilité avec l'environnement, et les problématiques de modélisation du risque. A partir de ce constat, il peut être souligné deux principaux constats qui limitent l'état de l'art :

- Le besoin de formaliser des risques (simulables) dans la modélisation des processus de l'entreprise, tout en conservant un langage accessible clair de façon à ne pas parasiter les modèles de processus de bases. Cet enjeu sera abordé par la réponse à la question Q1 en cherchant à séparer la modélisation des risques du model.
- Le besoin d'outils capables de communiquer avec des composants externes, sans être invasif dans l'interface utilisateur devra être traité par les réponses aux questions Q2 et Q3 de ce manuscrit.

Les problématiques d'externalisation et d'interopérabilité des composants hétérogènes sont des concepts que la multi-simulation semble pouvoir gérer.

3.La multi-Simulation à événements discrets

Cette section présente les solutions existantes dans le domaine de la multi-simulation. Dans la littérature, le concept de multi-simulation est principalement associé aux standards de co-simulation. Ici, nous y incluons également les standards de simulation distribué. Nous verrons dans un premier temps la théorie de la simulation distribuée, puis une alternative européenne plus récente : la co-simulation. Enfin nous verrons les principales implémentations de ces deux normes, leurs avantages et leurs inconvénients.

3.1.Théorie de la simulation distribuée

Les simulations à événements discrets de systèmes complexes deviennent problématiques du fait des limitations de la puissance de calcul sur les calculateurs monoprocesseurs. L'émergence de ces modèles complexes a entraîné l'apparition de technologies distribuées dans le domaine de la simulation. La simulation distribuée est une simulation exécutée par un ensemble de machines connectées à un réseau, ou un ensemble de processeurs sur une même machine. Ces différentes simulations font partie d'une simulation globale qui peut être considérée comme équivalente en termes de résultats à un seul et même simulateur [97]. Le terme « distribué » fait donc référence aux technologies qui permettent l'exécution d'un programme sur des systèmes informatiques multi processeurs, ou qui sont géographiquement distribués et connectés à un réseau [98].
Nous le verrons en détail plus tard dans ce manuscrit, les premières recherches sur la simulation distribuée ont été menées dans le domaine militaire. L'objectif principal était de parvenir à une réutilisation des modèles en résolvant des

problématiques d'interopérabilité entre des composants de simulation complexes, hétérogènes, et couteux. L'interopérabilité est un concept clé de la simulation distribuée, selon le centre technique et d'information de la défense américaine [99]. C'est « la capacité d'un modèle de simulation à fournir des services à d'autres modèles, à accepter des services d'autres modèles, et à utiliser les services ainsi échangés pour leur permettre de fonctionner efficacement ensemble ». Au-delà des capacités d'interopérabilité et de réutilisabilité, les principes de la simulation distribuée offrent d'autres avantages [98] :

- En répartissant les charges de travail sur plusieurs machines, on fragmente la complexité sur différents processeurs, ce qui implique une réduction du temps de calcul ;
- L'exécution de la simulation sur un ensemble de machines réparties géographiquement permet à de multiples participants physiquement situés sur des sites différents de communiquer comme s'ils étaient au même endroit ;
- Cette dispersion géographique permet donc une intégration de simulateurs provenant de différents fournisseurs. Plutôt que d'intégrer des programmes hétérogènes sur une application, il peut être plus rentable de les connecter via un mécanisme de simulation distribuée ;
- L'utilisation de plusieurs machines en parallèle augmente la tolérance aux défaillances et aux erreurs. Comme dans les problématiques d'équilibre de charges des serveurs, en cas de panne d'une machine, les autres membres du réseau peuvent se partager le travail délesté par la machine hors service.

Une simulation distribuée est par définition constituée de plusieurs composants, chacun étant responsable d'un sous-ensemble qui une fois reliés, constituent le système dans sa globalité [100]. Chaque composant d'une simulation distribuée dispose d'une horloge de simulation locale que l'on nomme Temps Virtuel Local (ou Local Virtual Time : LVT). Cette horloge contrôle le temps du composant distribué pendant l'exécution de la simulation.
Pour que des sous-systèmes fassent parti d'un système, les composants doivent être synchronisés et interconnectés dans le but de partager des informations en respectant la causalité dans le temps. Cette interconnexion passe par des canaux d'entrée/sortie fonctionnant comme des files d'attente de messages. Toutes les interactions entre les composants sont des messages datés chronologiquement.
Ce mécanisme de canal communiquant implique une subtilité : la liaison entre deux composants distribués peut être unidirectionnelle, ou bidirectionnelle, ce qui implique une potentielle dépendance entre plusieurs entités. Dans le cas d'une liaison unidirectionnelle, un premier composant produit une donnée qui est transmise et réceptionnée par le second. Cette donnée est attendue, puis consommée. On distingue donc une dépendance du second composant envers le premier. Il n'y a pas de retour d'information possible, ils peuvent donc être exécutés indépendamment. Dans le cas d'une communication bidirectionnelle, les deux composants ont tous les deux un canal d'entrée et de sortie, ils produisent donc tout deux des données, et sont capables d'en recevoir. Ils ne peuvent donc pas être exécutés indépendamment car les événements futurs de chacun d'entre eux pourraient dépendre des événements générés par l'autre.

Chaque machine ayant la responsabilité d'exécuter un composant de la simulation distribuée, elles avancent à une vitesse contrainte par la complexité des calculs à effectuer, et par la qualité des périphériques qui la composent. On peut donc en déduire que dans un contexte distribué, chaque sous-processus est exécuté à une vitesse différente des autres. En partant du principe qu'un évènement (ou message) externe peut apparaitre (en provenance d'un composant connexe) à n'importe quel moment de la simulation, on s'expose à des problématiques de causalité illustrés Figure 16, tiré de [101]. Le principe de causalité affirme que la cause d'un évènement précède son effet.

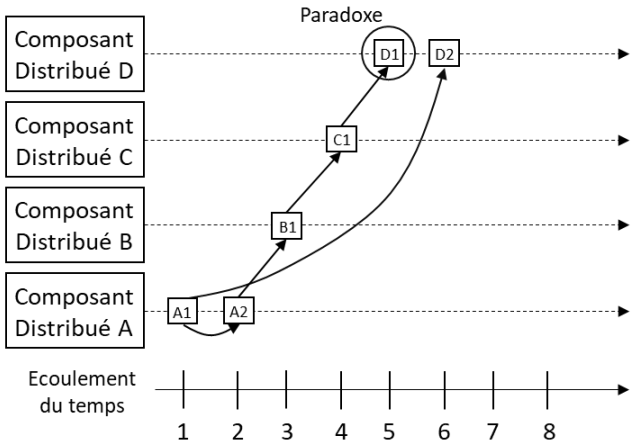


Figure 16 : Paradoxe temporelle dû à une violation du principe de causalité

La Figure 16 illustre un paradoxe temporel qui peut se produire en cas de violation du principe de causalité énoncé précédemment. L'exemple part du principe que 4 composants distribués (A, B, C, et D) font partie de la même simulation et sont donc délocalisés sur des machines différentes, chacune ayant son propre temps local. Le scénario décrit en Figure 16 montre plusieurs événements transmis entre les différents composants au cours du temps.
Au démarrage de la simulation, le composant A reçoit l'événement A1 avec l'horodatage 1, le traitement associé à cet événement consiste à envoyer deux nouveaux événements : A2 avec un horodatage 2 et D2 avec un horodatage 6. À la réception de l'événement D2, puisque le composant D n'a pas d'événement avant le temps de simulation 6, il avance son horloge et traite l'événement. Le composant A traite ensuite l'événement A2 qui génère B1 au temps 3. B1 génère C1 au temps 4, puis C1 génère D1 au temps 5. Lorsque le composant D reçoit D1, il a déjà traité D2, ce qui signifie que D a reçu un événement du passé dans son présent, ce qui pourrait affecter le reste de la simulation.
Afin d'éviter ce genre de situation, il est donc nécessaire d'utiliser des mécanismes de verrous temporels pour synchroniser les horloges autour des événements causaux. Leslie Lamport définit dans [102], une méthode basée sur les horloges logiques locales de telle façon que, pour tout événement a et b, si a est arrivé avant b, le temps logique local a doit être strictement inférieur à celui de b. Pour ce faire, il existe deux approches de synchronisation : conservative (ou pessimiste) et optimiste.

L'approche pessimiste conserve une cohérence temporelle de la façon la plus simple possible : elle maintient en permanence un ordre correct des exécutions

d'événements. Cette approche est introduite en 1970 par Chandy, Misra, et Bryant dans [103], [104]. Selon ce mode de fonctionnement, la synchronisation est faite avec l'aide de messages horodatés servant à décrire les événements. Aucun message ne sera horodaté d'une valeur inférieure au temps logique, quitte à générer un blocage dans la simulation distribuée. Chaque composant distribué communique avec les autres par le biais de canaux. Il va s'agir pour chacun d'entre eux d'envoyer les messages et d'exécuter les événements par ordre croissant horodaté. A n'importe quel moment de la simulation, chaque composant a accès à un ensemble de messages qui détermine les prochains événements à court terme (en fonction de la valeur du lookhead). Un composant peut alors simuler ou non un événement s'il constate qu'un autre événement doit arriver avant.

L'approche optimiste s'oppose à la méthode précédente : on considère qu'aucune prévention de contrainte temporelle n'est effectuée, mais qu'en cas de violation de la loi de causalité, on la corrigera. Ce principe intitulé « Time Warp » est proposé pour la première fois en 1985 par D.R. Jefferson [105]. Dans cette approche, les composants distribués exécutent les événements les uns après les autres dans un ordre naturel local. En cas de d'erreur temporelle, un mécanisme de retour en arrière est exécuté (appelé « Rollback »). Afin de garantir cette fonctionnalité, le système doit régulièrement effectuer des sauvegardes de son état afin de pouvoir y revenir en cas de nécessité.
Cette dernière approche peut proposer des gains de performance à condition de ne pas effectuer de retours en arrière fréquemment, ces derniers étant chronophages pour les performances de simulation. On note également des propositions d'amélioration de cette méthode dans la littérature comme Gafni Anat le propose dans [106], en optimisant les rollback, en annulant le moins de messages possible, ou en conservant certains messages inchangés en cas de retour en arrière.

Nous venons d'évoquer les principes de base de la simulation distribuée. Afin de les employer dans des cas concrets, il est utile de disposer d'un standard et d'une implémentation de ces différentes règles. Un des standards les plus utilisés dans ce domaine est HLA (High Level Architecture).

3.2.Historique et évolution HLA

Le standard HLA prend ses sources en 1983 sous le nom de projet SIMNET. Soutenu par la DARPA (Defense Advanced Research Projects Agency), SIMNET a été le premier système distribué pour des applications et des simulations de réalité virtuelle. Ce projet était le fruit d'une collaboration entre BBN (Bolt, Beranek et Newman) qui avaient la charge de gérer les échanges de données du système distribué, et par Perceptronics qui était en charge des formations. Le système SIMNET était utilisé par l'armée américaine pour les entraînements car l'entraînement avec du matériel réel était trop dangereux et trop coûteux. Son objectif était d'aider les unités militaires à s'organiser et à combattre en équipe. Son rôle était la communication de plusieurs simulateurs en réseau, où chaque simulateur était autonome, avait son propre affichage, ses propres contrôles et ses

propres ressources. Les interactions et les échanges de messages entre les simulateurs se faisaient sur une base P2P (Peer to Peer) sans système central.

En 1993, ce projet a donné naissance au protocole DIS (Distributed Interactive Simulation) [107]. Son principal objectif était d'améliorer et d'étendre les fonctionnalités de SIMNET. La norme IEEE du DIS est toujours disponible de nos jours sous le nom IEEE 1278 [108]. Cette norme était un protocole de communication, elle consistait à envoyer et à recevoir des messages appelés PDU entre différents objets de simulation. Le protocole DIS décrit 27 PDU différents qui sont responsables de l'échange de données entre les objets de simulation [97]. Le DIS a réussi mais présente des défauts de réutilisation et d'extensibilité. Par conséquent, la DARPA a prévu de développer une nouvelle architecture appelée HLA.

Le développement du standard HLA est entrepris par l'US DoD (Department of Defense). En 2000, il est adopté par l'IEEE et nommé « HLA IEEE 1516 ». Puis, il est mis à jour en 2010 [93] afin d'inclure certaines améliorations : cette nouvelle version est connue sous le nom de « HLA Evolved ». Le développement d'une nouvelle version de HLA a commencé en janvier 2016 par le SISO et est actuellement en cours. Il devrait s'intituler HLA 1516-20XX (HLA 4).

3.2.1.HLA - Orchestration de simulation distribuée

3.2.1.1.Concepts généraux

Le standard HLA propose une architecture qui facilite la réutilisation de composants distribués en passant par une simplification des problématiques d'interopérabilité. Il décrit également un ensemble de services et des règles pour l'implémentation de ces composants. Cependant, il ne propose aucun langage (textuel ou graphique) pour décrire la chorégraphie de ses composants durant l'exécution de la simulation [46].

3.2.2.Le vocabulaire HLA

Comme on peut le voir sur la Figure 17, dans un système HLA, on appelle l'ensemble de la simulation une « fédération ». Une fédération est composée de plusieurs sous-simulation distribuées, appelées des « fédérés », ce sont les simulateurs qui seront répartis sur plusieurs machines ou plusieurs processeurs. HLA introduit également un composant tiers appelé Run-Time Infrastructure (ou RTI), qui interconnecte tous les fédérés et qui met à leur disposition un ensemble de services dans le but d'assurer le bon déroulement de la simulation. Enfin, le « Fédération Object Model » (ou FOM) est un fichier qui contient la description rigoureuse des données échangées dans la fédération. Il en existe une variante, le « Simulation Object Model » (SOM) qui fonctionne de la même manière que le FOM mais qui est attaché à un unique fédéré : le fichier contient une description des informations qui vont transiter par le fédéré.

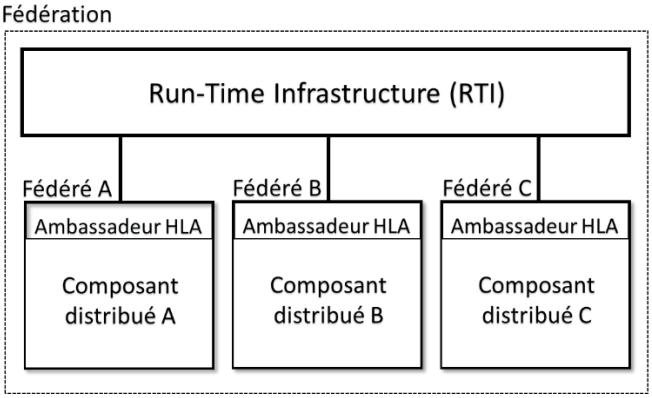


Figure 17 : Architecture HLA standard

3.2.2.1.Le standard HLA

La norme HLA est constituée de trois parties :

- Les « règles » au nombre de 10 pour les fédérés et la fédération sont fournies dans le document « HLA Rules » [93]
    - Règles pour la fédération :
        - Chaque fédération doit avoir un fichier FOM
        - Les fédérés doivent stocker les valeurs de leurs attributs, pas le RTI
        - Toutes les informations décrites dans le fichier FOM doivent transiter par les mécanismes de communication fournis par le RTI
        - Les fédérés doivent communiquer uniquement en employant les services fournis par le RTI (et ne pas communiquer par leurs propres moyens)
        - Chaque attribut de chaque instance d'objets doit appartenir à seulement 1 fédéré.
    - Règles pour les fédérés :
        - Chaque fédéré doit avec un fichier SOM
        - Chaque fédéré doit envoyer et recevoir des données telles qu'il est précisé dans le fichier SOM
        - Chaque fédéré pourra changer les propriétés d'objets tels qu'ils sont décrits dans le fichier SOM
        - Chaque fédéré doit envoyer et recevoir des données telles qu'il l'est décrit dans le fichier FOM
        - Chaque fédéré gère son propre temps local en utilisant les mécanismes de gestion du temps proposés par le RTI

Ces règles ont pour objectif de guider les développeurs à construire des fédérations réglementaires par rapport à la norme.

- La spécification des interfaces qui décrit en détails les services fournis par le RTI

La spécification des interfaces introduit les fonctions mises à dispositions des fédérés par le RTI et définit les spécifications d'interfaces entre le RTI et les fédérés :

- Management de la fédération : Ce groupe de fonctions permet de gérer la création, la modification et la destruction de l'instance de la fédération. On peut également grâce à ce groupe suivre l'évolution de l'exécution en temps réel de la simulation.
- Management de déclaration : Ce groupe fournit des services pour gérer les échanges de données entre les fédérés, en fonction des déclarations faites dans le(s) fichier(s) FOM. Ce service assure le suivi de chaque variable déclarée par les membres de la fédération. Selon HLA, une « variable » de communication peut être de nature différente : un *Objet* (persistant), ou une *Interaction* (éphémère). Une description avancée de ces principes est donnée dans le paragraphe suivant.
- Management de distribution des données : Ce groupe comprend les services qui gèrent la mise à jour des états et des informations échangées entre les simulations. Il limite et contrôle le volume de données échangées pendant la simulation entre les fédérés afin de ne transmettre que les données nécessaires à ceux qui en ont besoin.
- Management des objets : Gère la façon dont les fédérés peuvent instancier, mettre à jour, envoyer et recevoir des informations dans la fédération.
- Management des propriétés : Groupe de fonctions dont le but est de réglementer les différentes propriétés des objets dans la simulation.
- Management du temps : Ce groupe fournit des services liés à la gestion du temps dans la simulation. Cela regroupe des fonctions pour faire avancer le temps logique de la simulation ainsi que, la transmission des messages horodatés entre les fédérés.

Le Template de model objet (Object Model Template) décrit le format des données pour les fichiers FOM.
Il définit le format et la syntaxe (mais pas le contenu) des modèles d'objets HLA nécessaires à la réalisation des fichiers FOM et SOM. On rappelle que ces fichiers contiennent la description des moyens de communication entre les fédérés de la simulation.

3.2.2.2.Mécanismes de communications

Selon la norme HLA, la communication est fondée selon deux concepts, les Objets et les Intéractions :

- Communication via des Objets. Un objet est une instance persistante durant la simulation. Il se compose d'attributs ayant un nom et un type qui peuvent être mis à jour par un ou des composants. Un Objet est une instance représentant généralement une entité du monde réel. Cette structure de données est initialisée par un fédéré, et évolue au cours de la simulation. Les changements apportés à un Objet et ses attributs se font via la publication d'un fédéré aux attributs de l'Objet en question.

Réciproquement, le mécanisme d'abonnement (ou souscripition) permet d'être informé en cas d'évolution de l'Objet.

- Communication via des Interactions. Une interaction est une structure éphémère de la simulation, elle est utilisée comme un signal qui disparait après avoir été consommé. Une interaction se compose de paramètres ayant un nom et un type. Le déclenchement d'une interaction est de la responsabilité d'un fédéré.

Ces deux types d'informations sont échangés via un mécanisme commun de publication/souscription fourni par le RTI. Un fédéré peut enregistrer un objet puis modifier la valeur des attributs qui le composent. A chaque changement d'état, les fédérés ayant souscrit à cet objet recevront une mise à jour de la valeur.
De la même manière, un fédéré peut créer une interaction en précisant la valeur de ses paramètres. Tous les composants abonnés à cette interaction recevront une mise à jour de son changement d'état, la seule différence étant que l'interaction est supprimée de la simulation après sa création.

Il existe deux principales différences entre les Objets et les Intéractions :

- Lors de la mise à jour des attributs d'un Objet, le fédéré appliquant cette modification peut envoyer à la fédération un sous-ensemble des attributs de la classe, et non nécessairement tous les attributs. Par contre, pour les interactions, tous les paramètres sont nécessairement envoyés à la fédération.
- Pour toute classe d'Objet, un fédéré doit créer une instance de la structure de données. A l'inverse, la création d'une Intéraction ne nécessite pas d'instanciation de classe puisqu'il s'agit d'une information volatile.

3.2.2.3.Le temps selon HLA

Dans le standard HLA, les mécanismes de gestion du temps sont chargés de contrôler l'avancement des horloges des fédérés. Chaque composant distribué étant exécuté sur une machine locale, la notion du « temps présent » est relative.
Il existe différentes façons de gérer le temps dans HLA :

- Avancement par pas de temps : Chaque fédéré avance son propre temps par pas de temps fixe.
- Avancement basé sur les événements : Les fédérés avancent leur horloge à l'horodatage du dernier message qu'ils ont reçu (on rappelle que les messages peuvent être estampillés d'un temps que l'on appelle « Time-Stamp-Order » ou TSO).
- Avancement optimiste : Chaque fédéré est libre d'avancer son propre temps mais en cas de violation de la contrainte de causalité, on effectue un retour en arrière dans le temps.

Comme énoncé dans le chapitre précédent, les deux premières méthodes sont appelées méthodes « conservatives » [110], la dernière est appelée « optimiste »[111].
Les avancées temporelles sont coordonnées par les fonctions de management d'objet et permettent aux fédérés de se désigner comme étant « régulateur temporel » (TR pour « TimeRegulating ») ou « contraint temporellement » (TC pour « TimeConstrained »). Un fédéré régulateur peut contrôler la progression

temporelle logique des fédérés contraints sachant qu'un fédéré peut à la fois être TR et TC. Tous les fédérés régulateurs (TR et TR/TS) sont responsables de la régulation de l'avancement des fédérés TC. Pour garantir qu'un fédéré régulateur n'enverra aucun message TSO pendant une période donnée, les fédérés TR ont une valeur d'anticipation : le lookhead. Chaque fédéré déclare une valeur de lookahead afin d'éviter d'être trop en avance par rapport aux autres. Un TSO envoyé par un fédéré doit avoir son horodatage supérieur ou égal à l'heure locale du fédéré + sa valeur de lookahead.

3.2.2.4.Implémentations

Dans le cadre de HLA, les services de simulation distribués sont fournis par le RTI. Il existe différentes implémentations de ce RTI : certaines open sources comme Portico [112], ou CERTI [113], d'autres sous licences commerciales comme VT/MAK [114] ou Pitch [115]. Dans ce manuscrit nous travaillerons à partir de la librairie Pitch pRTI. L'utilisation de l'implémentation du RTI fourni par Pitch Technologies offre différents avantages comme l'accès à certains outils (Visual OMT qui apporte une aide à la conception des fichiers FOM), une documentation d'utilisation / tutoriels, et l'accès à une large communauté d'utilisateurs.

3.2.2.5.Discussions et conclusion

Le standard HLA est largement utilisé dans le domaine de la simulation distribuée. On peut cependant lui concéder deux principaux points faibles évoqués dans la littérature :

- Sa difficulté d'implémentation et d'utilisation.

Selon [116], HLA est un outil de simulation énergivore, chronophage, nécessitant beaucoup d'investissement et des compétences informatiques pour être mis en place et utilisé, ce qui rend l'utilisation du standard lourde et couteuse. Pour parer à cette défaillance, Falcone et al. proposent dans [117] un Framework de développement HLA permettant de mettre au point plus rapidement et plus efficacement des prototype HLA. Cette proposition est déployée au sein d'un événement annuel ayant pour objectif de former et de promouvoir la simulation distribuée : Le projet Simulation Exploration Experience (SEE) [118].

- Son absence de langage graphique pour spécifier et décrire le fonctionnement du standard ou ses interactions.

Dans [119], Falcone et al. proposent l'utilisation du standard BPMN pour modéliser et mieux comprendre le fonctionnement interne du standard. Leur approche se divise en deux parties : la première vise à utiliser le standard pour modéliser les mécanismes de fonctionnement internes d'un fédéré ; La seconde vise à utiliser le standard pour modéliser les différentes phases du cycle de vie d'une fédération HLA.

Dans [92], les auteurs présentent un Framework collaboratif baptisé CBPM (Collaborative BPM) basé sur du Business Process Management (BPM), ainsi qu'un logiciel intermédiaire qui met en œuvre les fonctionnalités du CBPM par l'utilisation de fonctions de base de la norme HLA. Le CBPM permet de définir et de simuler des moteurs de workflow utilisant HLA.

Taylor et al. présentent dans [121] des propositions d'amélioration d'un outil de simulation commerciale (Commercial-off-the-shelf simulation packages CSPs) en l'ouvrant à la simulation distribuée. Les auteurs proposent de mettre en communication des modèles de processus commerciaux déjà existant (modélisé en BPMN) en passant par un réseau de fédérés HLA.
Bocciarelli et al. ont présenté dans [122] une méthode de transformation automatisée de processus BPMN en modèles EQN (Extended Queuing Network). Les modèles EQN sont ensuite exécutés comme des simulations distribuées dans un scénario de commerce électronique. Ces travaux de recherche convertissent le BPMN en code exécutable.
Bazoun et al. ont présenté dans [123] un outil nommé « SLMToolBOX » qui est un modeleur graphique, un transformateur de modèle et un moteur de simulation capable de convertir un diagramme BPMN en modèles DEVS pour simuler le comportement d'un processus. Ces modèles DEVS peuvent être transformés et exécutés comme composants d'une fédérations HLA [124].
Enfin, Lee présente dans [125] un système de simulation de workflow qui combine à la fois les fonctionnalités de BPMN et de HLA. Le système en question permet aux experts de décrire un système industriel sous forme de diagramme BPMN. Le modèle ainsi obtenu sera converti en un ensemble de composants HLA qui seront simulés grâce à l'utilisation d'un middleware HLA-BPMN.

Après analyse de l'état de l'art relatif au standard HLA, certaines limites peuvent être identifiées. Selon la norme, les fédérés membres d'une fédération sont autonomes. Dans une démarche de réutilisabilité, ils doivent être retravaillés (redéfinir la structure, le code, les paramètres, les protocoles d'échanges, etc.), ce qui peut poser problème dans un environnement à forte réutilisation des composants. Dans la littérature, certains articles ont déjà souligné les bienfaits qu'apporterait un langage de modélisation graphique à la norme HLA. Malgré cette constatation, on remarque que peu d'études ont tenté d'utiliser un langage graphique dans le but d'orchestrer une simulation distribuée HLA. Représenter la séquence d'exécution de fédéré et leurs interactions par un langage graphique permettrait l'augmentation du potentiel de réutilisabilité et modularité du standard.

Les travaux exposés précédemment permettent via diverses méthodes d'exécuter des langages de workflow distribués tel que BPMN en passant par le standard HLA. Certain utilisent des standards graphiques tel que BPMN pour représenter les mécanismes internes de simulation et d'interaction de HLA. Cependant, on peut remarquer le besoin d'orchestration de simulation distribuée orchestré par un modèle graphique. Un des objectifs de ce manuscrit (au travers de la question Q3) est d'utiliser un langage graphique proche du BPMN pour désigner des composants de simulation distribuée, afin d'obtenir d'une part, un diagramme représentatif du scénario de simulation, et d'autre part, un moyen simple et graphique d'éditer l'orchestration d'une simulation distribuée.

Dans le domaine de la simulation distribuée, des alternatives existent. Plus employé dans le domaine industriel pour sa simplicité d'implémentation, le standard Functional Mockup Interface (FMI) propose une alternative à l'interconnexion de simulations.

3.3.Functional Mockup Interface (FMI)

3.3.1.Alternative à HLA : la Co-simulation

La simulation distribuée est un paradigme permettant de modéliser des systèmes distribués dynamiques et hétérogènes. Son but est non seulement d'accélérer les simulations, mais aussi de permettre l'implémentation de technologies stratégiques afin de relier différents types de composants de simulation [126].

Il existe plusieurs approches pour résoudre les problématiques d'interopérabilité dans le domaine de la multi-simulation. Deux des efforts les plus populaires allant dans ces directions, sont le standard HLA (High Level Architecture) que nous venons d'évoquer, et le standard FMI (Functional Mock-up Interface) [127].

FMI (Functional Mock-Up Interface) est une norme européenne conçue en 2011 par MODELISAR [128], [129] pour améliorer la conception des systèmes et logiciels embarqués dans les véhicules. Le standard est pensé et développé pour des applications industrielles, notamment pour des systèmes cyber-physiques, avec pour but de faciliter les échanges. Un de ses objectifs est de simplifier la collaboration entre des partenaires industriels, en leur donnant un moyen d'échanger des modèles, tout en garantissant une protection des secrets industriels.

Aujourd'hui, le standard supporte deux interfaces pour deux utilisations possibles : l'échange de modèles (Model Exchange : ME) et la co-simulation (CO) [130].
FMI propose un ensemble de règles génériques et un ensemble de fonctions pour manipuler des composants spécifiques : des FMU (Functional Mock-up Unit).
Un FMU peut donc être :

- Un composant destiné à de la co-simulation
- Un composant destiné à l'échange de modèles

Ces composants à importer sont vus comme des boites noires contenant du code compilé (des librairies dynamiques). Un FMU peut donc être donné à un collaborateur qui pourra l'exploiter au sein d'une co-simulation en employant l'interface d'interaction spécifiée par la norme. Le couplage des simulateurs par FMI CO masque les détails de leur mise en œuvre et peut donc protéger la propriété intellectuelle.

Le standard fournit des interfaces entre le décideur et les exécutants pour traiter à la fois les questions d'échange de données, et d'algorithmes, de façon continue ou discrète [130]–[132]. L'interface se divise en deux parties : une utilisation pour le couplage d'outils ou de simulation, et une pour le couplage de modèles de systèmes et sous-systèmes. Nous avons donc deux types de FMU :

- FMU pour l'échange de modèles (ME) Figure 18 : il contient un modèle ainsi que sa structure, sans solveur. Le composant accueillant le FMU doit

implémenter lui-même un moteur d'exécution qui doit faire évoluer les variables d'état du modèle.

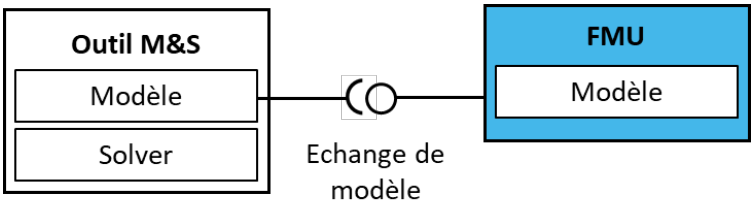


Figure 18 : FMI pour l'échange de modèle

L'interface FMI pour l'échange de modèles définit une interface (voir Figure 18) avec le modèle d'un système dynamique décrit par des équations différentielles, algébriques et en temps discret. De plus, il fournit une interface pour évaluer des équations selon les besoins dans différents environnements de simulation, ainsi que dans les systèmes de contrôle embarqués. L'interface est conçue pour permettre la description des grands modèles.

- FMU pour les Co-Simulation (CS). Il contient un modèle, sa structure, ainsi qu'un solveur pour le faire évoluer au cours du temps. C'est le moteur intégré au FMU qui fera évoluer les variables d'état du composant. Dans cette configuration, seules certaines variables d'entrée pourront être modifiables par l'utilisateur.

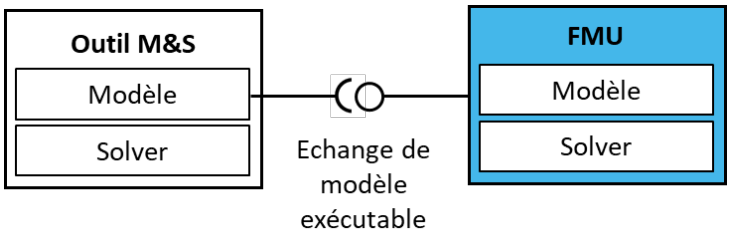


Figure 19 : FMI pour la co-simulation

Un composant FMU est en réalité un fichier compressé contenant :

- Une interface de co-simulation : un ensemble de fonctions C pour le contrôle des exécutants et l'échange de données d'entrée et de sortie ainsi que d'informations d'état compilées en bibliothèques.
- Un schéma de description de la co-simulation : il définit la structure et le contenu du fédéré via un fichier XML. Ce fichier XML spécifique à l'exécutant, contient des informations « statiques » sur le modèle (variables d'entrée et de sortie, paramètres, ...) et le solveur/simulateur (capacités, ...). Des « flag » caractérisent la capacité de l'exécutant à prendre en charge des algorithmes décideurs avancés qui utilisent des pas de communication variables, une extrapolation de signal d'ordre supérieur, etc. Il est à noter que le standard FMI ne peut prendre en charge que des variables standards.
- Il peut contenir des fichiers sources et header du FMU (pas obligatoire).

Le point intéressant dans cette architecture n'est pas l'utilisation d'un fichier XML pour transmettre des informations à une bibliothèque logicielle, mais les méthodes d'interaction à notre disposition pour communiquer avec les FMU. FMI pour la co-simulation est capable de gérer les routines de communication entre le décideur (composant qui charge le FMU) et l'exécutant (le FMU). Chaque exécutant possède un ensemble de fonctions pour communiquer avec le décideur, et inversement, dans la limite des variables de type simple. Il est intéressant de constater que le décideur est responsable de l'avancement du temps de chaque FMU, et qu'aucun algorithme de gestion du temps n'est proposé par le standard. Ces deux derniers points représentent des facteurs clé pour l'articulation de cette norme avec d'autres standards de multi-simulation.

FMI est un standard permettant l'interconnexion de nombreux composants compressés dans un format particulier. Il résout les problématiques d'interopérabilité à condition que les modules adoptent son formalisme. De nos jours, la complexisté des systèmes étant croissante, les besoins techniques et les métiers vont au-delà des standards de modélisation et de simulation. On observe divers travaux de recherche qui tendent à créer des ponts entre le standard FMI et son environnement.

3.3.2.Capacité d'interopérabilité de FMI

On observe dans la littérature différents auteurs tentant des approches pour réduire ou résoudre les problématiques d'interopérabilité propres au standard de co-simulation notamment FMI. Certains d'entre eux tentent de créer des rapprochements entre FMI et HLA comme Yilmaz et al. qui présentent dans leur article [133] une méthode pour générer automatiquement un fédéré à partir d'un FMU. Un outil lit les spécifications d'un fichier FMU et génère une enveloppe HLA autour de ce dernier, afin de le rendre connectable à un RTI. Une démonstration de cette proposition (appelée FMUFd pour FMU-federate) est faite avec l'implémentation HLA proposée par VT MAK.
Dans l'article, les auteurs sont toutefois restreints par les types de données utilisées par FMI.

Dans le cadre de la simulation distribuée et des problématiques d'interopérabilité de systèmes cyber physiques, Lasnier et al. proposent dans [134] de combiner un outil de modélisation et simulation des systèmes hétérogènes (Ptolemy II) avec le standard HLA. Les auteurs proposent le développement d'une extension de l'outil afin de permettre la communication avec des composants externes en passant par HLA. Au travers de cette implémentation, les auteurs se focalisent sur les mécanismes de gestion du temps entre les composants fédérés, et les performances d'exécution d'un système de commandes de vol.

Franke et al. proposent dans [135] l'implémentation de mécanismes d'événements discrets dans le standard FMI compatibles avec les caractéristiques synchrones de Modelica et Dymola. Leur contribution se divise en deux parties :

- L'ajout d'une horloge dans la description de modèles FMU

- Une extension de l'interface d'appel des fonctions permettant l'activation de ces horloges par des composants externes.

Cette proposition permet la synchronisation des FMU avec des composants de l'environnement Modelica ou Dymola.

Dans [136], Garro et al. présentent deux solutions possibles pour intégrer des entités FMU dans une fédération HLA :

- HLA for FMI : Un FMU contenant les libraires de communication HLA lui permettant de se connecter à une fédération en tant que fédéré
- FMI for HLA : Un fédéré HLA agissant comme master d'une (ou plusieurs) composants FMU. Le master a la responsabilité de gérer le temps et les différents échanges entre la fédération et les différents FMU exécutants.

Ces deux méthodes permettent une communication entre les deux standards, mais les problématiques fondamentales aux standards restent inchangés :

- Incompatibilité des types complexes
- Systèmes de communications trop éloignés
- Mécanismes d'avancement du temps différents

Le standard FMI répond donc en partie à notre besoin d'interopérabilité des composants hétérogènes industriels car il permet l'interconnexion d'outils industriels variés. Cependant, il ne permet pas d'évoluer dans une simulation distribuée synchronisée compatible HLA. Nous allons devoir proposer dans le chapitre 4 d'allier ces deux approches pour répondre à la problématique en développant les travaux de Garro et en adaptant notre solution à notre contexte industriel. Cela devra passer dans un premier temps par une preuve de concept en déployant un FMU au sein d'une simulation distribuée HLA afin d'apporter un premier élément de réponse à la question Q2. Dans un second temps, nous proposerons d'adapter cette proposition au sein de notre outil de M&S Papyrus.

4.Synthèse et articulation des contributions

Comme nous l'avons vu au cours de ce chapitre, le contexte industriel et les problématiques balayent plusieurs champs de recherche : la gestion des risques, la modélisation, la simulation, la multi-simulation. Nous venons d'analyser l'état de l'art de ces différents domaines pour nous permettre de dégager des pistes de répondre à la problématique formulée au travers de plusieurs interrogations.

Dans la suite de ce manuscrit, nous tenterons d'apporter des réponses structurées en trois chapitres qui formeront les briques d'une architecture englobant l'ensemble des propositions (voir Figure 20). Les différentes contributions sont délimitées par les cadres pointillés et détaillées ci-après.

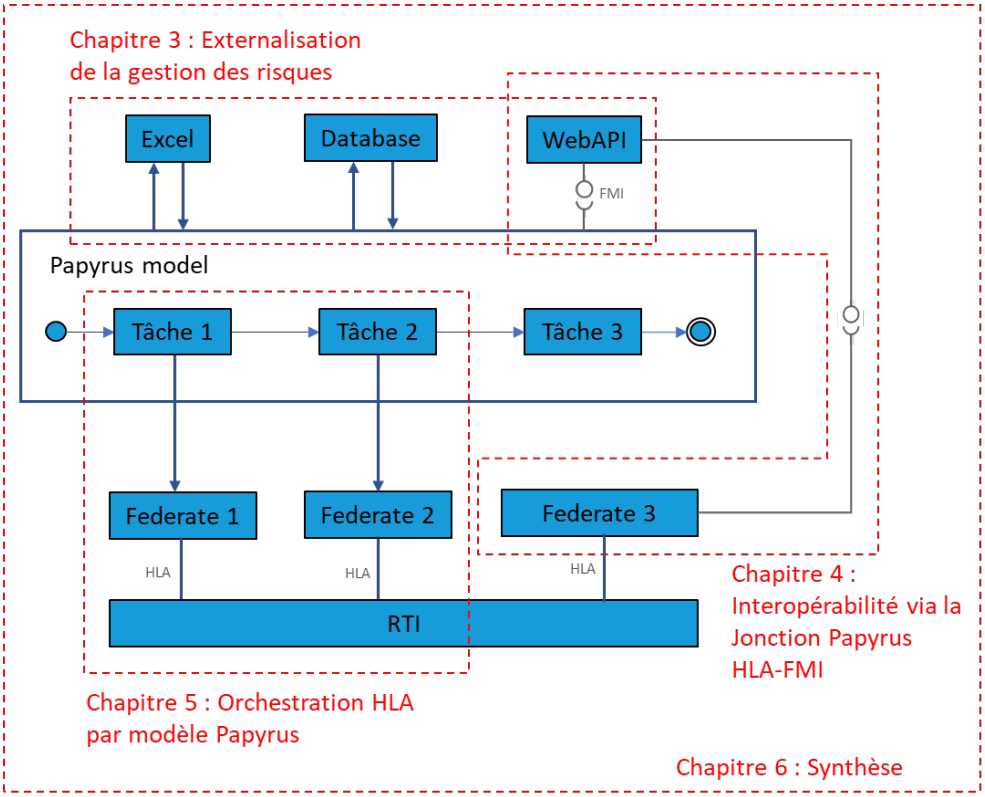


Figure 20 : Architecture globale

Le chapitre 3 de ce manuscrit devra aborder les problématiques de gestion des risques au travers de l'outil de modélisation et simulation Papyrus (Q1) en complément de l'état de l'art. Nous verrons dans ce premier chapitre de contribution le fonctionnement de Papyrus et comment l'étendre pour gérer des mécanismes de générations d'aléas en dehors d'un modèle de simulation. Cela nous permettra de connecter l'outil à des éléments externes tels que des tables Excel ou des bases de données.

Le chapitre 4 contribuera à répondre à deux problématiques d'interopérabilité présentes dans le standard de co-simulation FMI. La première concerne la compatibilité avec le standard HLA, ou comment faire communiquer un fédéré avec un composant FMU (Q2.1). La seconde concerne l'utilisation de FMI pour la co-simulation au sein de l'environnement Papyrus pour effectuer des requêtes via des outils extérieurs (Q2.2).

Le chapitre 5 s'intéressera dans un premier temps à la définition d'un fédéré « décideur » capable de piloter des fédérés « exécutants » par l'implémentation d'une surcouche logiciel au standard HLA (répondant ainsi à la question Q3.1). Par la suite, cette proposition devra être implémentée en une extension Papyrus permettant de modéliser un scénario de simulation distribué. L'outil de modélisation Papyrus sera étendu afin de désigner des fédérés, le moteur d'exécution papyrus sera un fédéré décideur de pilotage de simulation distribuée (répondant ainsi à la question Q3.2).

Enfin, le chapitre 6 de ce document aura pour objectif de regrouper les propositions décrites dans les chapitres de contributions. L'idée est de proposer un

environnement de modélisation et de simulation hétérogène distribué et dirigé par un processus orchestrateur, tout en disposant d'une gestion des risques et aléas externalisés. L'outil proposé dans ce chapitre baptisé DIAMANT++ devra hériter de toutes les extensions présentées au cours de ce manuscrit et pourra être utilisé dans le cadre d'un cas d'application inspiré du contexte d'ALSOLENTECH

Chapitre 3
Externalisation de la gestion des risques de Papyrus

Sommaire

1.Introduction

L'objectif de ce chapitre s'articule autour de la gestion des risques dans un environnement industriel de simulation. Les étapes de modélisation et de simulation sont des étapes obligatoires dans le processus de conception de n'importe quel système complexe. Cela permet d'anticiper et d'étudier le comportement et les interactions d'un sujet avec son environnement. Les systèmes devenant de plus en plus volumineux et complexes, la difficulté de simulation et les risques liés augmentent proportionnellement. Parallèlement à cette complexité croissante, les risques, les dangers et les menaces doivent également être pris en compte lors du processus de modélisation.

Dans ce chapitre, nous proposons la séparation des données d'entrée d'un modèle de simulation, et la gestion des risques qui les concernent illustré par la Figure 21. Cette proposition permet d'apporter des modifications à un modèle nominal sans surcharger le modèle de base.

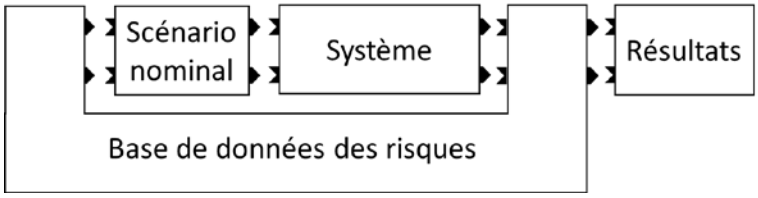


Figure 21 : Séparation entre les données d'entrée d'une simulation et la gestion des risques

Autrement dit, nous allons délocaliser la modélisation des risques en dehors du scénario nominal pour réduire la complexité, et augmenter la portabilité du composant de simulation.

2.Architecture de Papyrus

Durant ce chapitre, nous proposons de concevoir un outil permettant la modélisation d'un système dans son état nominal, la modélisation des risques qui lui sont associés, et la simulation de cet ensemble. Pour ce faire, nous rappelons que les risques vont être délocalisés dans un outil en dehors de Papyrus permettant d'obtenir une séparation distincte entre le modèle, et l'outil de gestion des risques.

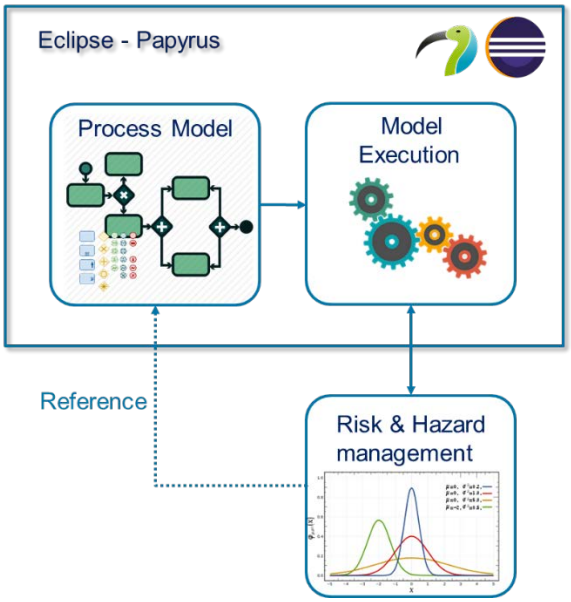


Figure 22 : Externalisation des risques de Papyrus

La Figure 22 illustre la proposition de ce chapitre. On peut y observer le bloc « Eclipse – Papyrus » contenant le plugin de modélisation (Process Model) et le plugin de simulation (Model Exécution). Notre proposition consiste en la création d'un module externe de gestion et génération de risque et aléas, connecté au moteur d'exécution de Papyrus. Ce dernier, capable de référencer les tâches du model primitif nominal, permet d'agréger des données supplémentaires au modèle de base et ainsi influencer le déroulement de la simulation sans polluer la représentation des processus.

3.Référencement des activités pour la gestion des risques

L'objectif étant d'altérer le déroulement d'une simulation Papyrus, le module de gestion des risques doit être capable d'identifier un processus ou une action particulière du modèle afin d'injecter une perturbation temporelle (voir Figure 23). Cela va passer par plusieurs étapes que nous étudierons en suivant :

- Identifier un composant du modèle de base par un identifiant unique.
- Modifier les attributs temporels du composant, au moment de la phase de simulation du modèle.
- Définir un outil externe à Papyrus permettant de créer un couple probabilité/impact qui sera affecté sur une ou plusieurs tâches.
- Conserver une trace des perturbations générées par l'outil pour assurer un traçage et un historique de la simulation.

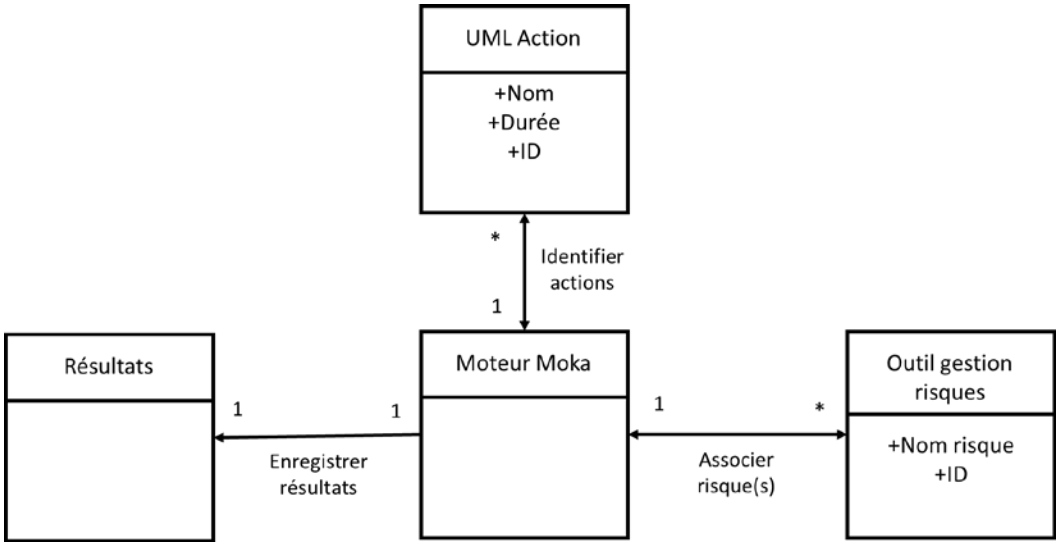


Figure 23 : Architecture UML haut niveau des objectifs du chapitre

La Figure 23 représente l'architecture macroscopique UML de notre proposition. Notre objectif est d'affecter les actions UML du modèle Papyrus depuis l'outil de gestion des risques en passant par le moteur Moka. Pour ce faire, il est important d'être capable de différencier les tâches entre elles. C'est la raison pour laquelle nous ajoutons la notion d'identifiants unique à chaque tâche. Comme on peut le voir sur la figure, une action UML (qui représente la brique de base des modèles Papyrus) est pourvue par défaut d'un nom et d'une durée. Nous ajoutons un champ d'indentification (ID) pour la différencier des autres éléments. Cet identifiant est défini par l'utilisateur lors de la modélisation. En parallèle, l'outil de gestion des risques a la fonction de définir des lois impactant une ou plusieurs tâches UML au moment de la simulation. La connexion entre une loi et une action nécessite également la déclaration d'un identifiant unique (ID) dans l'outil de gestion des risques. Durant la simulation, le moteur d'exécution Moka inscrit les résultats dans une base de données représentée par le bloc « Résultats ».

Afin d'ajouter cette notion d'identifiant unique, nous devons définir un nouveau profil UML

4.Profil UML

Selon le standard UML [34], un profil est un moyen de fournir un mécanisme d'extension générique permettant de personnaliser les modèles UML de base. Il permet de personnaliser le langage graphique et sémantique afin de le rapprocher d'un contexte particulier. Les ajouts faits sur le langage de base ne peuvent contredire la sémantique du langage standard originel.
Les profils sont définis par le biais de « stéréotypes » que l'on applique à des éléments du langage de base (comme les classes, les attributs, etc.). Ils sont donc un ensemble d'extensions qui personnalisent la norme pour un contexte précis. Ex : Avant d'être des standards autonomes de l'OMG, le SysML et BPMN étaient des profils UML.

Comme nous venons de l'évoquer, nous devons ajouter un paramètre aux tâches exécutées sous Papyrus afin de leurs attribuer un identifiant unique. Les tâches UML exécutées étant des Actions UML, on applique donc le stéréotype de la Figure 24 dans le but d'ajouter la notion d'identifiant à chaque tâche Papyrus.

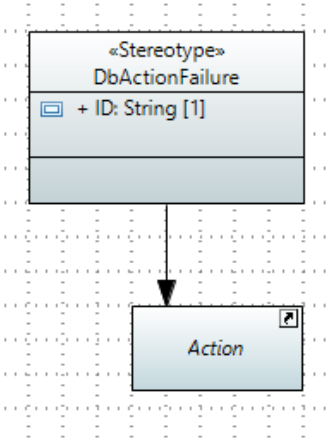


Figure 24 : Profil UML pour enrichir le modèle d'un identifiant

Ici, on applique un stéréotype aux actions UML ce qui permet de les enrichir d'un paramètre « ID ». Le profil ainsi créée (FailureProfile) ajoute aux actions UML un stéréotype *DbActionFailure* attribuant ainsi l'identifiant. Sur la Figure 25, en selectionnant la tâche « Site exploitation », on peut visualiser l'ensemble des profiles appliqués à l'action UML. On applique donc le *FailureProfile* à la tâche, ce qui permet ainsi de l'enrichir d'un l'identifiant.

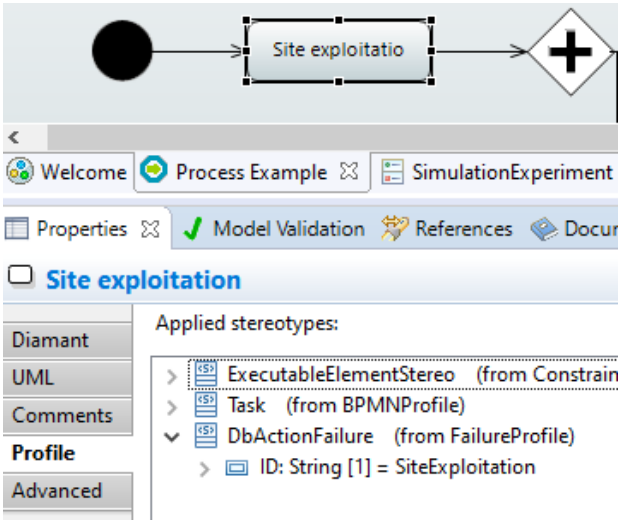


Figure 25 : Application du profil UML dans Papyrus

Maintenant qu'une tâche peut être identifiée, le moteur d'exécution doit être en mesure d'interpréter sa valeur et d'impacter cette tâche par un risque externe.

5.Gestion du temps dans Moka

Moka est un moteur d'exécution basé sur la gestion des événements. A chaque événement, un ensemble de composants vont être interrogés afin d'autoriser ou non l'exécution d'une tâche. Ces composants sont appelés des *Visiteurs* et des *Advices*. Dans le fonctionnement de Moka, un Visiteur est instancié à chaque

élément du diagramme UML de base : points de départ, d'arrêt, transitions, étapes, etc. Comme on peut le voir sur la Figure 26, chacun de ces Visiteurs contient une liste d'Advices qui lui sont associés. Chaque Advice permet d'ajouter un comportement supplémentaire au moteur.

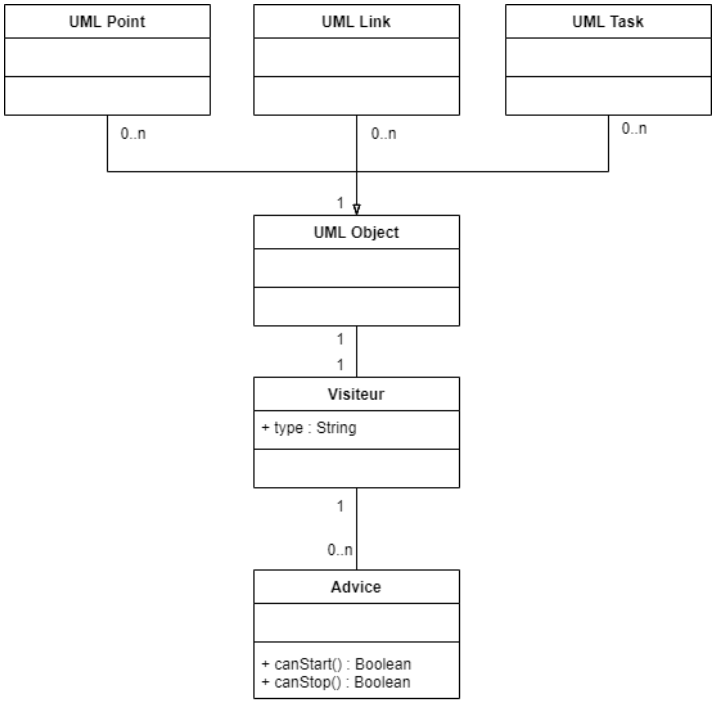


Figure 26 : Diagramme de classe UML des Visiteurs et Advices

Pendant le déroulement d'une simulation, chacun des Visiteurs va être consulté par Moka au moment où sa tâche est active. A la consultation d'un Visiteur, l'ensemble des Advices associés au Visiteur sont exécutés afin d'exécuter le comportement qu'ils décrivent. La Figure 27 représente le système d'avancement dans le temps du moteur Moka au cours d'une simulation.

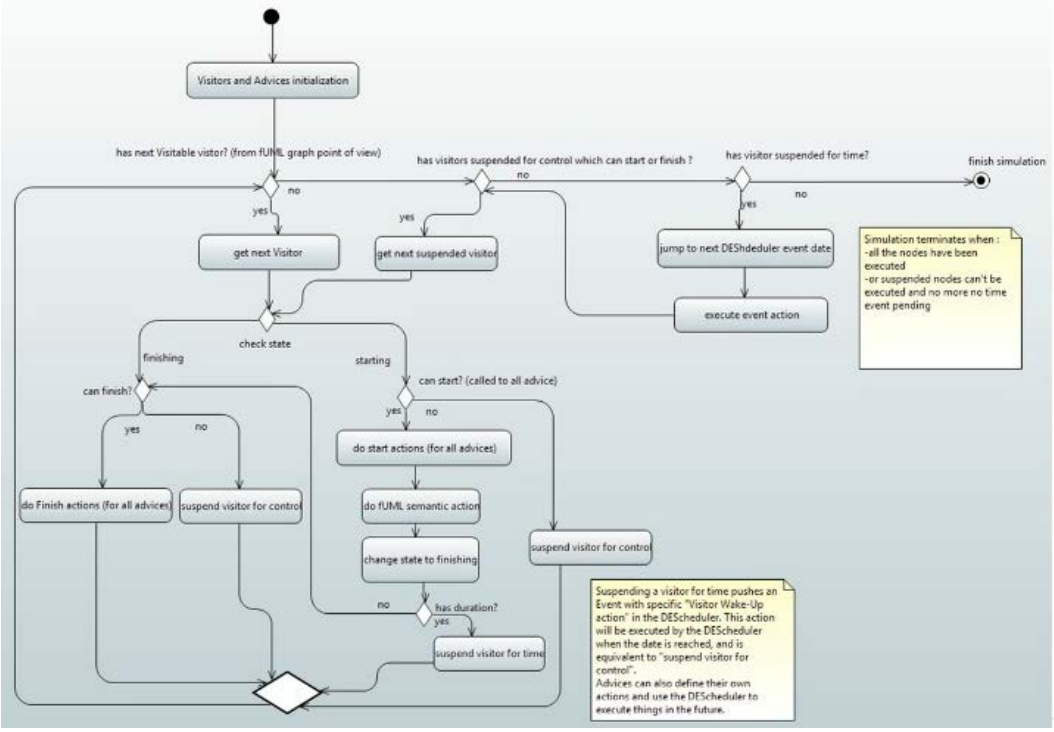


Figure 27 : Processus d'exécution d'un diagramme UML par Moka

L'algorigramme de la Figure 27 est un bouclage itératif que le moteur exécute durant une simulation.

Dans la première étape de la Figure 27, le moteur récupère tous les Visiteurs présents dans le modèle. L'étape suivante est une boucle conditionnelle : pour chacun des Visiteurs, on va regarder s'ils sont « visitables ». Un Visiteur est « visitable » s'il n'a pas d'étape précédente (début de simulation du modèle), ou si son étape précédente est terminée. Si on peut consulter un Visiteur, on le charge en mémoire et on va regarder pour chacun de ses Advices s'ils autorisent à exécuter le Visiteur.
Durant la simulation, un Advice associé à un Visiteur va intervenir à 2 moments : au démarrage de l'action et à la fin de l'action. En effet, il va pouvoir empêcher de démarrer ou de finir une action.
Pour un Visiteur, on va regarder son état qui peut être « starting » : s'apprête à démarrer, ou « finishing » : s'apprête à se terminer.
Dans le cas d'un démarrage, on boucle sur tous les Advices associés au Visiteur. L'interrogation de la fonction *canStart()* permet de demander l'autorisation d'exécuter le Visiteur. Nous appelons cette fonction pour chaque Advice. Si tous les Advices répondent favorablement, on exécute les fonctions *start()* des Advices. Puis, la tâche est exécutée par le moteur. Elle n'est pas terminée mais mise dans l'état « finishing » : elle va attendre une autorisation de se terminer. Si une durée a été associée à la tâche, elle va être placée dans une file d'attente liée au temps. Le Visiteur sera réveillé plus tard par le DEScheduller du moteur Moka (mis dans la pile « suspended for time »). Si la tâche n'a pas de contrainte de temps, le processus d'exécution tente de terminer la tâche. On tente de la faire se terminer en interrogeant les fonctions *canFinish()* des Advices associés au Visiteur. Si la terminaison d'une tâche est acceptée, on appelle les fonctions *finish()* des Advices,

et on passe au Visiteur suivant. Si on ne peut pas terminer la tâche, c'est qu'elle est bloquée par un autre Advice d'un autre Visiteur, le moteur place donc ce Visiteur dans une pile (pile nommée « suspended for control »), la tâche courage est donc bloquée en exécution.
Une fois que tous les Visiteurs visitables sont terminés, Moka va reprendre les Visiteurs stockés dans la pile « suspended for control ». En effet, les Visiteurs en cours d'exécution n'ayant pas le droit de se terminer ont peut-être été débloqués par les fonctions start et finish des Advices précédents. On effectue donc les mêmes actions pour les Visiteurs de cette pile. Puis, on vérifie les Visiteurs suspendus pour le temps (liste « suspended for time »). Ces Visiteurs sont classés par date d'exécution. Quand le moteur arrive à cette liste, aucun autre Visiteur ne peut être exécuté. Il avance donc dans le temps et saute à l'évènement du Visiteur le plus proche, et l'exécute.

Du point de vue du moteur d'exécution Moka, on peut donc représenter les interactions entre le modèle de base, et ses différents Visiteurs, comme sur la Figure 28.

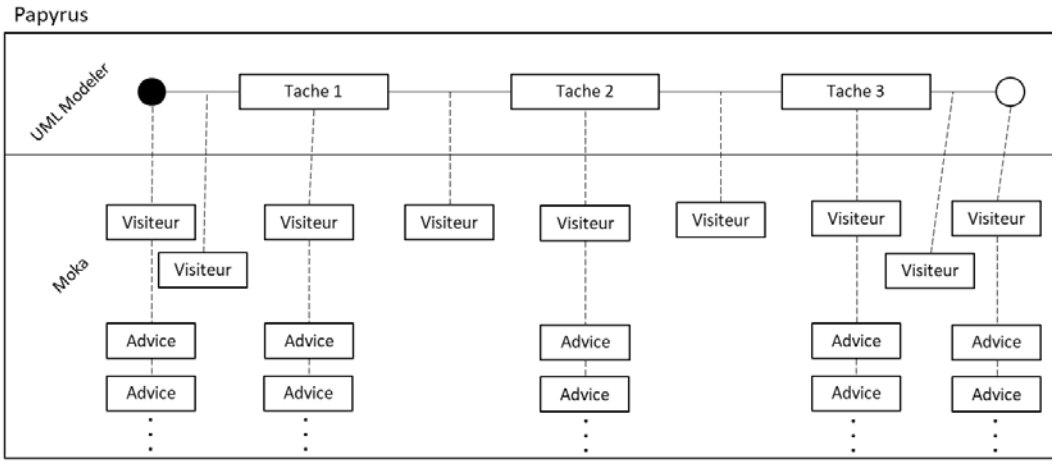


Figure 28 : Visiteurs et Advices connectés à un diagramme UML

Du point de vue du modèle Papyrus (partie haute de la Figure 28), on constate que chaque élément du diagramme UML est lié à son Visiteur associé. Chacun des visiteurs de la simulation peut avoir aucun, un, ou plusieurs Advice(s) associés.

Il nous est donc possible d'étendre le moteur d'exécution de Papyrus pour ajouter des Advices aux Visiteurs, et donc ajouter des conditions / comportements personnalisés lors de l'exécution d'un modèle Papyrus.

6.Gestion du temps au travers des profils UML

Dans Papyrus, le temps d'exécution d'une tâche est géré par un profil UML *ConstrainedExecutionProfile* (visible sur la Figure 29). Ce profil permet d'ajouter la notion de temporalité à une action UML, et donc d'être simulée par fUML. En d'autres termes, ce profil permet d'ajouter à une tâche, la notion de durée sous la forme d'une chaine de caractère (« duration : String » sur la Figure 29), ainsi qu'un traceur de résultats sous la forme d'un booléen (« isLogged : Boolean »).

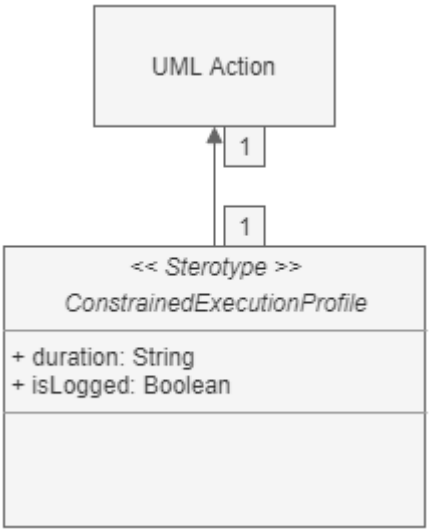


Figure 29 : Profil UML : « ConstrainedExecutionProfile »

Le paramètre « duration » détermine la durée d'exécution d'une tâche sous la forme d'une chaine de caractère : « hh mm ss », où h représente les heures, m les minutes, et s les secondes. Ce paramètre est modifiable par l'utilisateur lors de la modélisation afin qu'il puisse déterminer un temps d'exécution pour chaque tâche du modèle. Ce paramètre sera transmis au moteur dans la phase de simulation.
Le paramètre « isLogged » est un ajout fourni par le CEATech qui permet de déterminer si la durée d'une tâche sera enregistrée en bases de données ou non. Si cette variable est vraie, alors à chaque changement d'état, cette tâche sera horodatée dans une base de données influxDB et visualisable sur une interface graphique. Dans le cas contraire, l'état de la tâche ne sera pas tracé. Ce profil UML est associé automatiquement à chaque tâche lors de leur création.

7.Création d'une extension Moka pour l'externalisation des risques

Une fois le profil UML défini, il est nécessaire de développer une extension du moteur Moka afin d'interagir avec les paramètres ajoutés à notre modèle UML. Avec l'outil « EMF Generator Model » inclus dans Papyrus, nous avons développé une extension du moteur d'exécution de Papyrus (visible sur la Figure 30) permettant d'interpréter les données contenues dans le profile UML de gestion des risques.

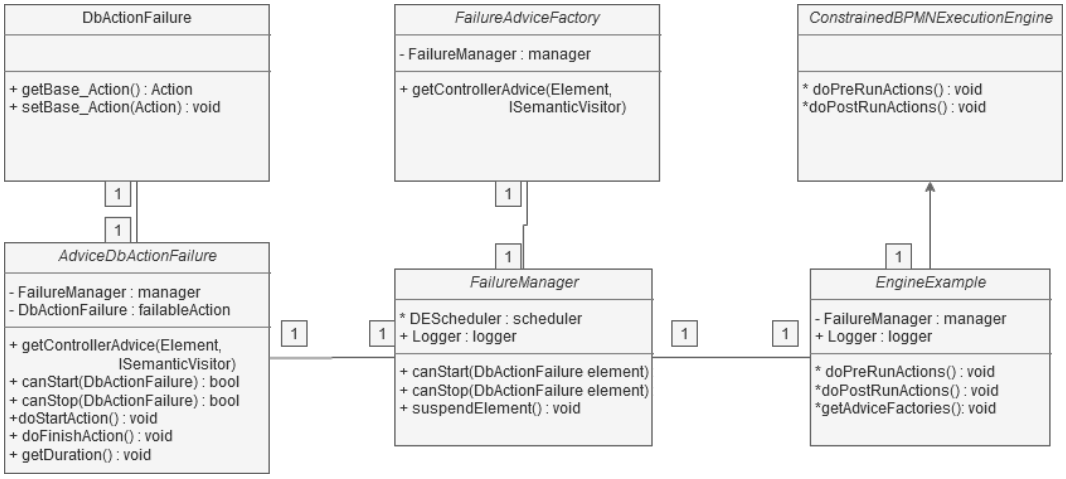


Figure 30 : Diagramme de classe UML du code généré par l'EMF generator

Le diagramme de classe UML de la Figure 30 est l'implémentation d'une extension du moteur d'exécution Moka. Il nous permet de définir le comportement du moteur lors de l'exécution d'un stéréotype appliqué à une tâche UML précise.

Le point d'entrée de ce diagramme est l'*EngineExample*. Il est référencé directement par Moka en héritant du moteur d'exécution de Papyrus. Nous nous intéressons ici à 3 fonctions provenant du moteur que nous surchargeons pour adapter le moteur à notre contexte.

- doPreRunActions : est une fonction lancée au démarrage d'une simulation. Cette fonction est lancée avant de parcourir le graphe conçu par l'utilisateur et avant d'instancier les Visiteurs et leurs Advices. C'est via cette fonction que nous initialisons les classes permettant la gestion des risques au moment de la simulation : la classe *FailureManager* ainsi que les différentes variables nécessaires au fonctionnement de l'outil (systèmes de gestion des logs, interface utilisateur, etc.)
- doPostRunActions : est lancée en fin de simulation. En mode « production », l'outil collecte l'historique des actions effectuées par la simulation et plus particulièrement les actions effectuées par la couche de gestion des risques afin d'en suivre l'activité. En mode « développement », cette fonction est utilisée pour collecter l'historique des actions effectuées ainsi que les éventuelles erreurs et exceptions rencontrées durant la simulation.
- getAdviceFactories : est une fonction appelée au démarrage du moteur. Elle va récupérer la liste des Advices instanciée par défaut par le moteur. Nous surchargeons cette fonction afin d'ajouter effet cumulatif le(s) Advice(s) personnalisés (dans notre cas, *FailureAdviceFactory*).

La classe FailureManager implémente les fonctions appelées par le moteur pour autoriser ou non l'exécution d'une tache : *canStart()*, *canStop()*, et *suspendElement()* qui permet de suspendre la durée d'une tâche.

La classe FailureAdviceFactory est appelée pour chaque tâche UML à l'initialisation du processus de simulation (voir première tâche de la Figure 27). Elle parcourt l'ensemble des éléments UML et permet d'associer (ou non) un Advice

customisé au Visiteur de la tâche courante. Dans notre cas, cette classe va associer un « AdviceDbActionFailure » pour chaque Action UML étendue par le profil « FailureProfile ». La classe implémente la fonction :

- *getControllerAdvice(UMLElement, Visiteur sémantique)* qui prend en paramètre un élément UML (un arc, une tâche, etc) de notre diagramme, ainsi que le Visiteur sémantique à associé à l'élément UML. C'est sur ce Visiteur que l'on va greffer notre nouvel Advice. La greffe d'un Advice sur un Visiteur va se faire en retournant une nouvelle instance d'AdviceDbActionFailure.

La classe *AdviceDbActionFailure* représente un Advice. Il est appelé par le Visiteur d'une tâche UML et contient les fonctions :

- *canStart()* : qui est appelée par le moteur. Retourne un booléen pour autoriser ou non l'exécution de la tâche. C'est à partir de ce point que l'on va lancer le failureManager (*manager.canStart()*) pour demander l'autorisation d'exécution d'une tâche.
- *canStop()* : appelée par le moteur pour demander l'autorisation de finir cette tâche (il se réfère lui aussi au failureManager).
- *doStartAction()* : est appelée quand la tâche est autorisée à démarrer.
- *doFinishAction()* : est appelée quand la tâche va se terminer.
- *getDuration()* : détermine la durée d'exécution de la tâche imposée par l'Advice courant. C'est via cette fonction que nous introduirons la dégradation temporelle d'une tâche comme expliqué dans la partie suivante.

8.Impact d'un risque : dégradation de la durée d'exécution d'une tâche

La conséquence d'un risque étant d'entraîner la dégradation de la durée d'une ou plusieurs tâches, il est nécessaire que cette perturbation agisse sur les durées au cours de la simulation. Pour avoir un outil le plus flexible possible, l'utilisateur définit chacun des risques présents dans la simulation. Il y a donc une phase de modélisation des risques à entreprendre en parallèle à la modélisation des processus. Le recensement des risques peut se faire de différentes manières en fonction du contexte. Nous détaillerons cette partie plus tard, dans les applications.

Un risque peut avoir deux fonctions cumulables :

- Il peut être un élément référencé par un identifiant unique pouvant avoir un impact sur la durée d'une ou plusieurs tâches d'un processus Papyrus. Cet impact peut être une constante, le résultat d'une équation, ou basé sur un facteur semi-aléatoire.
- Il peut aussi être un élément aggravant l'impact d'un autre risque de la simulation.

Nous avons déjà explicité le fonctionnement de l'association tâche / risque par le biais des identifiants, nous allons maintenant décrire comment augmenter la durée d'une tâche via un Advice.

Dans Papyrus, la durée d'une tâche UML est calculée par effet cumulatif de l'ensemble des Advices associés à la tâche. Comme on peut le voir sur la Figure 31, pour un élément UML est associé un Visiteur qui contient un ou plusieurs Advice(s). Le Visiteur appelle les méthodes *getDuration()* de tous ses Advices et somme leurs résultats.

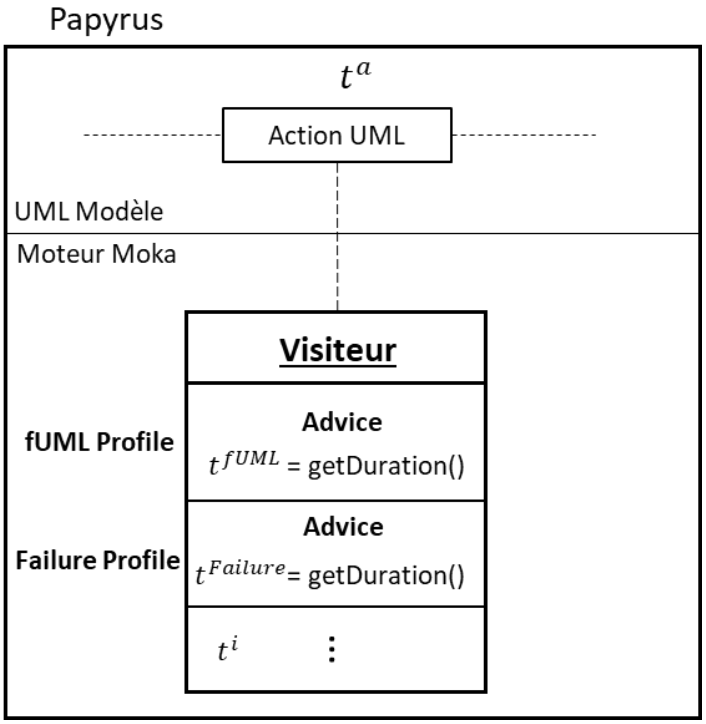


Figure 31 : Détermination de la durée d'une tâche via les Profils

La méthode retourne un double représentant une durée en nanosecondes. Par conséquent, la somme des durées des Advices représente le temps total de la tâche (t) sous la forme de l'Équation 1 :

$$t^a = t^{fUML} + t^{Failure} + \sum_{i=0}^{n} t^i$$

Équation 1 : Calcul du temps de simulation dans Moka

C'est via des instances de l'Advice *AdviceDbActionFailure* et la méthode surchargée *getDuration()* que l'on dégrade le temps associé à une tâche. La prise en charge d'une action UML via l'Advice est explicitée dans le code de la Figure 32.

```
1  Double getDuration () {
2          […]
3          String taskName = failableAction.getBase_Action().getName();
4          //recuperation de l'identifiant de la tâche
5          String duration = "1m" ;
6          //définition d'une valeur de temps par défaut
7          String actualDate = getCurrentDate("dd/MM/yyy HH:mm:ss");
8          //récupération de la date courante
9          failureManager.setFailureDate(actualDate, failableAction.getID())
10 ;
11         //on écrit la date courante et on lance le calcul
12         duration =
13 failureMananger.getTaskDuration(failableAction.getID()) ;
14         //récupération des résultats
15
```

```
16
17          if(duration == null || !DurationHelper.isValdString(duration)) {
18          //verification format du temps
19                  duration = “1m”;
20                  failureManager.logger.log(failableAction.getId() + “ :
21  error.                  No string or formatError”);
22          } else {
23                  failureManager.logger.log(failableAction.getID() + “ : “ +
24                          duration found in tool = “ + duration);
25                  //log
26          }
27
28          long durationInMs = DurationHelper.getDurationNanos(duration);
            //conversion en Nanosecondes et retour
            return (double) durationInMs;
    }
```

Figure 32 : Fonction getDuration() de l'Advice

Lors de l’exécution d’une tâche étendue par le profil *FailureProfile*, on récupère l’identifiant de la tâche UML (l.9, l.11) ainsi que son nom (l.3) via son instance (méthodes *getBase_Action()* et *getID()* de la classe *DbActionFailure* Figure 30). Cet identifiant est transmis à l’outil de management du risque (l.9) afin de lui indiquer :

- L’identifiant de la tâche en train d’être exécutée par le moteur de simulation.
- La valeur de l’horloge de Moka au moment où cet Advice est interrogé.

Une fois ces informations transmises, le *failureManager* est interrogé pour connaître quel va être l’impact des risques sur l’action en cours. Cette valeur sera retournée par la fonction *getTaskDuration* sous la forme d’une chaîne de caractère (l.11).
La gestion des dates et durées se fait sous la forme de chaînes de caractères pour une meilleure lisibilité et une simplicité de configuration par l’utilisateur final. Cette fonctionnalité implique une vérification de la validité de la syntaxe (l.14). De plus, une gestion des historiques de simulation est mise en place via des fichiers log implémentés dans le *failureManager*. Enfin, la durée supplémentaire générée par l’Advice est convertie en nanosecondes (l.25), puis renvoyée au moteur (l.27).

Le processus décrit par le code Figure 32 est exécutée pour chaque *FailureAdvice* instancié par le moteur (donc pour chaque tâche UML étendue par le profil *FailureProfile*). Comme on peut le constater sur la Figure 33, quand une tâche UML est étendue par un *FailureProfile*, le moteur ajoute automatiquement à son Visiteur l’Advice de gestion des risques (*AdviceDbActionFailure*).

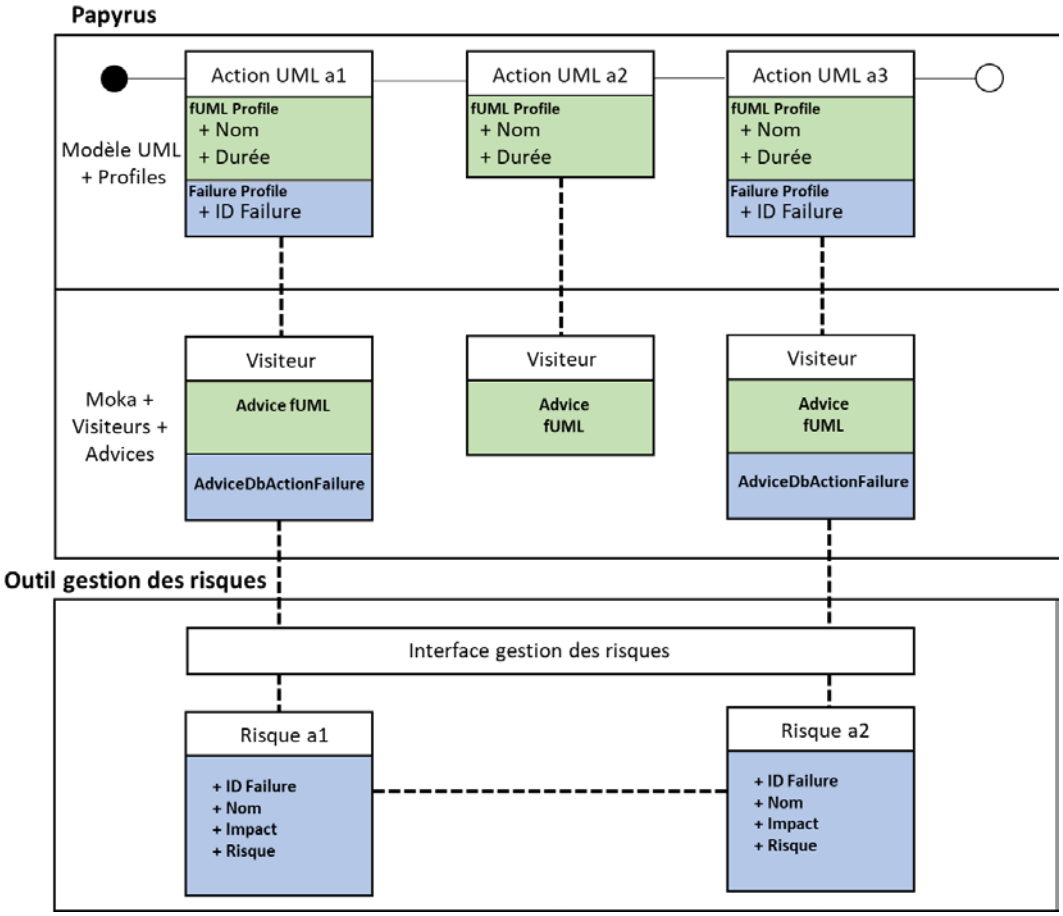


Figure 33 : Papyrus + outil de gestion des risques : Advices et leurs interactions extérieures

Par la suite, l'Advice effectue une requête à l'outil de management du risque au moment de l'exécution de son action UML associée. On remarque que sur la Figure 33, chaque Action UML possède son aléa associé et déporté dans l'outil de management. Un *AdviceDbActionFailure* récupère l'identifiant « ID Failure » présent dans l'action UML (par le biais du *FailureProfile*), et interroge l'interface de gestion des risques pour obtenir les informations de l'aléa ayant le même identifiant. Dans le cas de la Figure 33, « ID Failure » du *FailureProfile* doit être strictement égale à « ID Failure » de « Risque a1 » dans l'outil de gestion des risques. Il est à noter que deux risques peuvent être liés entre eux : ils peuvent avoir des liens de causes à effets.

L'outil de management du risque doit donc contenir une liste exhaustive de risque inhérents au modèle représenté sous Papyrus. De plus, il est absolument obligatoire que chaque identifiant indiqué par le *FailureProfile* fasse référence à un identifiant de risque dans l'outil déporté. En cas d'oubli de la part de l'utilisateur, aucun risque ne sera pris en compte pour la tâche La Figure 34 représente le cycle d'exécution du modèle présenté en Figure 33 : le cycle de vie d'une simulation connectée à l'outil de gestion des risques.

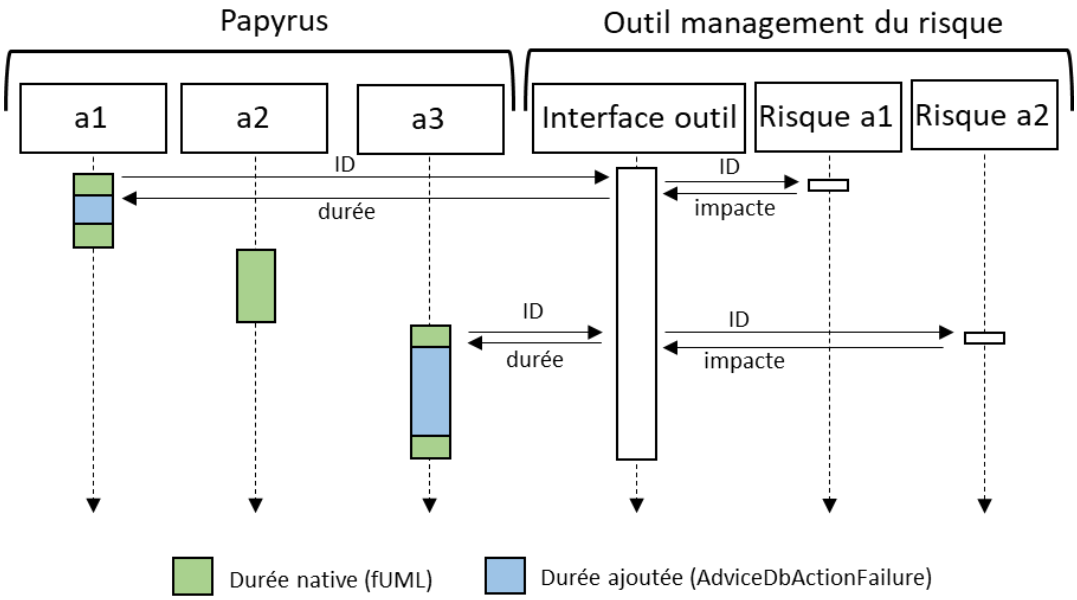


Figure 34 : Diagramme séquence UML de gestion des risques et Papyrus

Conformément au modèle de l'architecture Figure 33, les trois tâches a1, a2 et a3 se succèdent. Lors de l'exécution de la première tâche, Moka interroge le Visiteur de a1, qui lui-même interroge les différents Advices qui le composent :

- L'Advice fUML définit la durée par défaut de la tâche (le temps vert sur la Figure 34)
- *L'AdviceDbActionFailure* récupère l'identifiant du *FailureProfile* pour le transmettre à l'interface de l'outil de management du risque.

L'outil met à jours ses tables de données et retourne une réponse à l'Advice. Cette durée est transmise au moteur Moka qui simule ce temps (visible en bleu sur la Figure 34). La tâche a2 est ensuite exécutée, Moka interroge le Visiteur associé à cette tâche qui récupère le temps de son unique Advice : la durée verte. Enfin, la tâche a3 est elle aussi composée de l'Advice fUML et *AdviceDbActionFailure*, et son temps est donc le résultat de la somme des deux Advices : durée verte + bleue.

Nous venons de décrire le fonctionnement de Papyrus et comment étendre ce dernier afin de le personnaliser. Cela passe par l'extension de son outil de modélisation pour l'enrichir sémantiquement et, ainsi, se rapprocher des problématiques métiers. Cela passe également par l'évolution de son moteur d'exécution afin de permettre des comportements en lien étroits avec l'évolution sémantique de ses modèles. L'accès au code source de Moka nous permet d'influencer le cours d'une simulation avec des méthodes et outils extérieurs à Papyrus. Dans la suite de ce chapitre, nous verrons deux outils de management du risque développés pour fonctionner avec l'extension proposée précédemment. Le premier outil est conçu spécialement pour le cas d'étude ALSOLENTECH. Il répond à un besoin précis : l'utilisation d'Excel pour recenser les risques. La seconde version est une alternative à l'usage d'Excel : une base de données SQLite qui permet une plus grande flexibilité d'utilisation.

9.Applications : mise en œuvre sur deux outils

9.1.Application industrielle : dans le cadre d'ALSOLENTECH

Pour illustrer cette contribution, nous partirons d'un cas d'utilisation ALSOLENTECH. Le modèle représenté Figure 35 est conçu avec l'outil Diamant et représente le processus macro de mise en place d'une centrale électrique solaire.

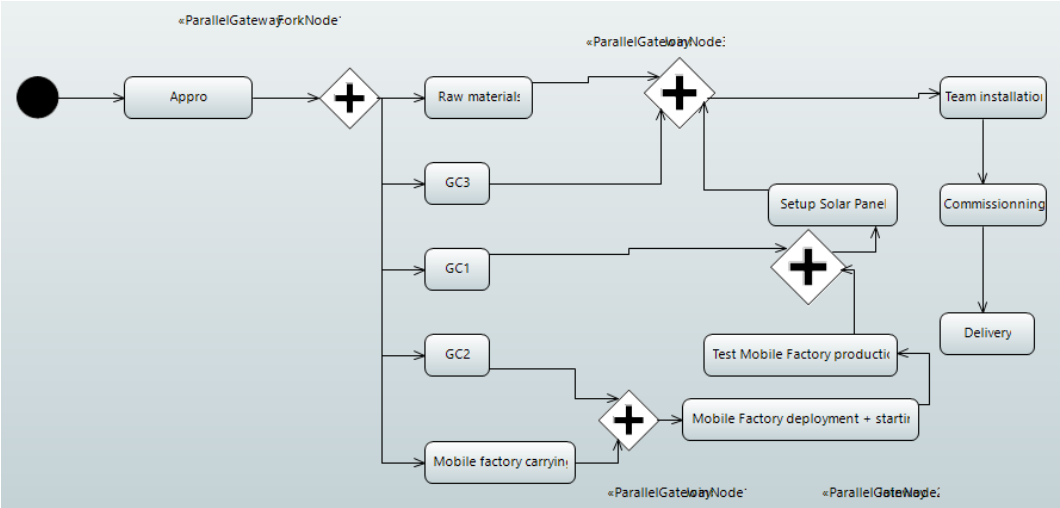


Figure 35 : Cas d'utilisation ALSOLENTECH - déploiement centrale

La première étape de ce modèle correspond à une simulation d'approvisionnement des matériaux nécessaires aux différentes constructions qui forment l'ensemble de la centrale électrique. Puis sont lancés parallèlement un ensemble de tâches :

- la livraison de matériaux nécessaires au fonctionnement de la centrale solaire : Raw materials ;
- la construction de 3 structures pour accueillir la centrale : Génie Civil (GC) 1, 2, et 3 ;
- la livraison de l'usine mobile sur le site de production : Mobile factory carrying.

Une fois l'usine mobile acheminée sur le site de production, et le génie civil 2 terminé, on peut simuler le déploiement et le fonctionnement de l'usine : Mobile « Factory deployment + starting » qui simule le déploiement de la centrale. Une fois les champs solaires déployés, une batterie de test sont effectués pour déterminer la puissance effective de la centrale : « Test mobile factory production ». Puis, s'enchaine la mise en production des panneaux solaires pour terminer par des dernières phases de tests avant la livraison de la centrale.

Chacune de ces étapes est soumise à de nombreux facteurs de risque :

- Certains risques dépendent d'outils externes que nous verrons plus tard dans les Chapitre 4 et 5 de ce manuscrit ;
- D'autres dépendent des bases de connaissances ou de l'expertise d'ALSOLENTECH.

C'est à partir de ce dernier point que nous adapterons notre proposition pour un contexte industriel.

Dans le cahier des charges fonctionnel, deux caractéristiques inhérentes à l'outil sont indispensables :

- Simplicité : Les outils et fonctionnalités développées pour répondre aux problématiques de l'entreprise doivent être utilisables par des personnes non-expertes de la programmation informatique ou des mécanismes avancés de simulations. La modélisation des processus est assurée par l'interface graphique de Papyrus et sa gestion des profils UML. La modélisation des risques est gérée via une extension Papyrus compatible avec Excel qui garantit une simplicité d'utilisation, une puissance et flexibilité des calculs, ainsi qu'une large communauté d'utilisateurs. Enfin, la simulation est assurée par Moka et son intégration dans Papyrus.
- Flexibilité : L'outil doit être en mesure de modéliser des systèmes (et leurs risques) quel que soit le domaine. L'utilisateur doit pouvoir basculer facilement d'un scénario à l'autre. Papyrus permet de charger des modèles via l'interface utilisateur hérité d'Eclipse. Nous présenterons également une interface afin de charger des scénarios de risques différents.

Afin de répondre à ces deux impératifs, la gestion et la génération des risques est pris en charge par un ou plusieurs fichiers Excel (voir Figure 36).

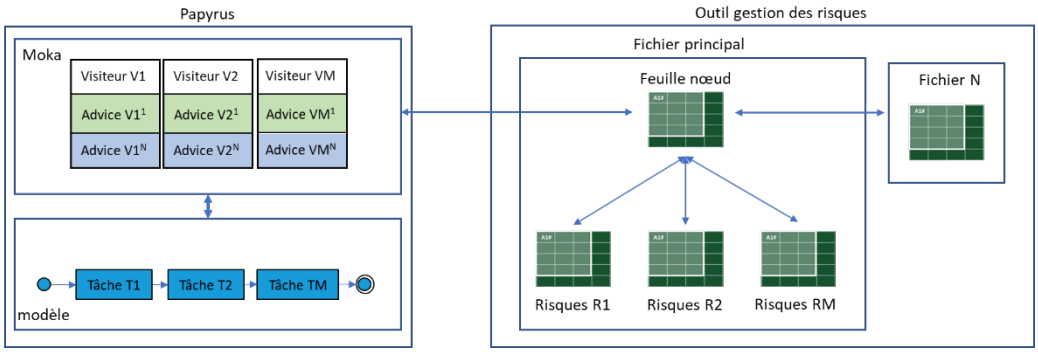


Figure 36 : Gestion des risques par feuilles Excel

Lors de l'exécution d'un modèle, Moka associe un Visiteur par tâche, chaque Visiteur a ses propres Advices. Nous proposons dans cette contribution un Advice customisée pour ALSOLENTECH conçu dans le but d'ajouter du temps de simulation. L'Advice récupère l'identifiant unique de la tâche et interroge le fichier Excel en lui fournissant la date actuelle afin de récupérer des informations sur les risques modélisés par l'utilisateur. L'ensemble de ces risques auront une répercussion sur le déroulement, et les résultats de la simulation Papyrus.

Afin de garantir une liaison fiable entre l'outil Papyrus et l'ensemble des feuilles Excel contenant les bases de connaissances industrielles, un fichier de configuration a été conçu pour définir un certain nombre de paramètres relatifs à l'interconnexion des deux outils. Ainsi, un ensemble d'étapes préliminaire à la simulation a été implémenté dans l'extension Moka. Lors du lancement d'une

simulation, une interface graphique interroge l'utilisateur sur le choix du fichier Excel à utiliser comme scénario de risque (voir Figure 37).

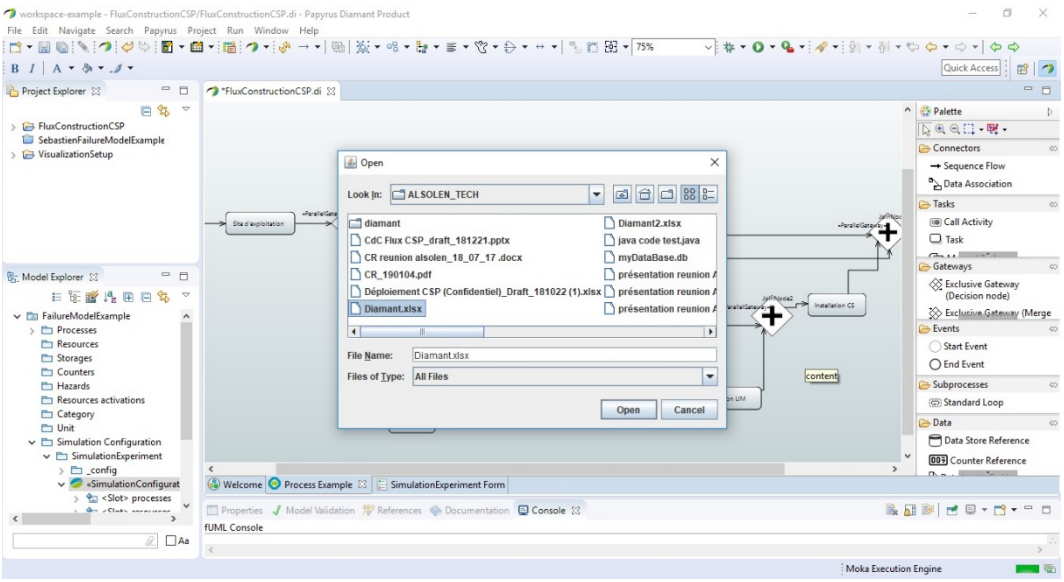


Figure 37 : IHM choix du fichier Excel de risques

Via cette fenêtre, l'utilisateur indique le chemin d'accès qui mène au fichier xlsx contenant les paramètres de risques relatifs au modèle Papyrus exécuté. De plus, il existe un mode de fonctionnement alternatif qui contourne le choix manuel systématique du fichier. Dans une démarche d'optimisation ergonomique, au lancement d'une simulation, le moteur interroge un fichier « config.json » contenant un ensemble de paramètres (voir Tableau 2 suivant) relatifs à la liaison du fichier Excel.

config.json	
mode_auto	enable / disable
targetFile	« emplacement du fichier »
targetDir	« emplacement du dossier par défaut »
sheetName	« nom de la feuille Excel interface »

Tableau 2 : Fichier de configuration relatif à la connexion Papyrus - Excel

L'état du paramètre « mode_auto » définira deux modes de fonctionnement :

- Un mode « développement » qui affiche l'interface de sélection du fichier Excel manuellement (visible sur la Figure 37) en pointant vers un répertoire par défaut précisé par le paramètre « targetDir ».
- Un mode « utilisation » qui charge automatiquement le fichier Excel indiqué par le paramètre « targetFile » sans demander de validation à l'utilisateur.

De plus, le nom de la feuille Excel qui servira d'interface entre les données de risques et Papyrus sera précisé par le champ « sheetName ».

Du point de vue du fichier Excel connecté à Papyrus, nous imposons un ensemble de règles afin de permettre une communication avec l'Advice de Papyrus. Les règles sont les suivantes :

- Le nom et chemin d'accès au fichier Excel de gestion des risques doit être précisé manuellement par l'interface, ou par la variable « targetDir » du fichier config.json

- Les données écrites et lues par Papyrus seront stockées dans la table « interface », qui portera le nom décrit par la variable « sheetName »
- La première colonne de la table interface est réservée à la lecture de Papyrus. On y écrira les valeurs ou formules Excel de dégradation du temps
- Les données temporelles décrites par la première colonne doivent respecter la syntaxe temporelle de Papyrus (1heure = 1h, 1seconde = 1s, etc.)
- La deuxième colonne de la table interface est réservée aux identifiants des tâches (ils font références aux identifiants uniques contenus dans le profil UML de gestion des risques)
- Un identifiant de la deuxième colonne doit être unique : il ne doit pas y avoir de doublons
- Dans la colonne des identifiants, le caractère « . » est réservé à la désignation de composants connexes à Papyrus. Ils permettent d'affecter des risques à des outils externes, que nous verrons dans le chapitre 6 de ce manuscrit.
- La troisième colonne de la table interface est réservée à l'écriture de Papyrus. Il viendra y écrire les dates à laquelle les tâches sont exécutées au format « jj/mm/aaaa hh:mm:ss »

L'ensemble de ces règles garantissent des échanges dynamiques entre Papyrus et Excel. Les deux contraintes relatives aux risques peuvent être statiques : définies en dur dans le tableau Excel, ou dynamiques : définies en fonction de formules Excel qui peuvent prendre en paramètres : des valeurs pseudo aléatoires, ou des dates de la simulation courante. Comme illustré sur la Figure 38, la communication avec le fichier Excel est horodatée.

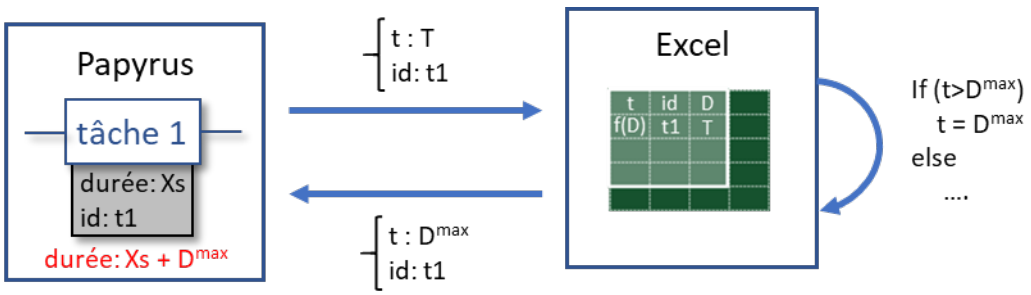


Figure 38 : Gestion des risques Excel dynamiques

Quand une tâche étendue par le profil de gestion des risques est exécutée, le moteur Moka interroge le fichier Excel en fournissant l'id de la tâche courante (id = t1), et la date actuelle de simulation (t=T). Cette dernière information est inscrite dans le tableau (voir règle relative à la colonne 3 de la table interface), puis l'ensemble des formules Excel sont mises à jours. La première colonne est donc actualisée, et récupérée par le moteur Moka. Dans le cas de la Figure 38, la durée de la tâche 1 est donc de Xs (sa durée initiale) + D^{max} (sa durée additionnelle relative aux risques encourus).

9.1.1.Exemple

Pour illustrer cette contribution, on part d'un exemple simplifié utilisé dans le cadre d'ALSOLENTECH (voir Figure 39) qui reprend le modèle présenté Figure 35. Nous rappelons que les processus et données décrits dans cet exemple sont modifiés et anonymisés pour des raisons de confidentialité. Le profil « DbActionFailure » est appliqué à la tâche « Raw materials » du modèle Papyrus. On observe alors l'apparition du champ ajouté par le nouveau profil de gestion des risques : « ID ».

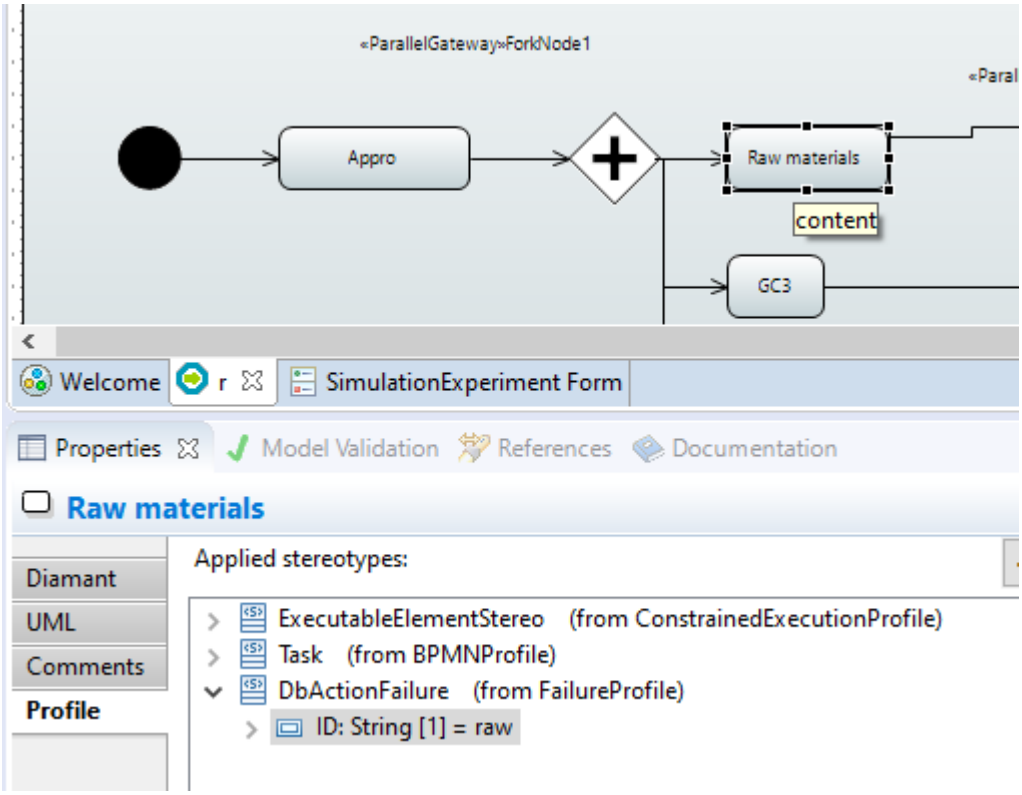


Figure 39 : Exemple interaction Excel

L'utilisateur définit un identifiant unique à cette tâche : « raw ». On rappelle que cet identifiant doit faire référence à une entrée du fichier Excel de risques (ligne 3 de la Figure 40). Dans un second temps, l'utilisateur doit décrire les ralentissements qui seront appliqués à cette tâche.

Données Diamant		
Durée	Identifiant	Date d'execution
277j	raw	03/07/2018 09:01:00
300j	GC1	
650h	GC2	

Input	
Fournisseur choisis	FOURN2
Surface GC1	100000

Figure 40 : Configuration fichier Excel

Ici, l'identifiant raw est lié à la formule de gauche (277h sur la Figure 40). Cette valeur est le résultat d'un calcul dépendant de trois facteurs :

- Le fournisseur choisit par l'entreprise : sélectionnable via la liste déroulante dans le menu « input » de la Figure 40.
- La date d'exécution de la tâche « raw » : qui sera complétée au moment de l'exécution de la tâche par Moka.

- Un nombre généré aléatoirement : reproduisible car le système de génération aléatoire est implémenté avec un système de seed.

ALSOLENTECH dispose d'une base de données de fournisseurs. Dans notre exemple, chacun d'entre eux est capable d'assurer une livraison avec un certain délai, à un certain coût, et dispose d'un niveau de confiance de l'entreprise. Ces informations sont stockés dans notre table (voir Figure 41) afin d'établir différents scénarios (risqué mais gros retour sur investissement, ou peu risqué mais faible retour sur investissement).

Règle appro equipement nominal				
Désignation équipement	**Réf. Fourniseurs**	**Prix (€)**	**Délai (j)**	**Confiance**
ORC	FOURN1	100	365	1
ORC	FOURN2	200	245	2

Règle appro : pertes / retard fournisseur			
FOURN1		FOURN2	
Probabilité	Impact retard (j)	Probabilité	Impact retard (j)
0,1	60	0,5	5
0,15	30	0,5	30
0,75	0		

Figure 41 : Tables de données des fournisseurs

Le tableau jaune est relatif aux règles d'approvisionnements : Le fournisseur 1 (FOURN1) possède le plus haut niveau de confiance, propose une livraison complète en 365 jours pour un prix de 100€. L'entreprise a également établi des lois de probabilités relatives aux retards de livraison : le fournisseur 1 a 75% de chance de livrer sans retard, 15% de chance de livrer avec environ 30 jours de retard, et 10% de chance de livrer avec environ 60 jours de retard.

Dans cet exemple, nous considérons également que les délais de livraison peuvent être affectés par le mois courant. Cette loi de probabilité est décrite dans la Figure 42.

Règle appro : facteur météo							
	Mai		Juin		Juillet		
	% Proba (0-1)	Impact	% Proba (0-1)	Impact	% Proba (0-1)	Impact	%
.0	0,01	90	0,9	2	1	0	
0	0,19	40	0,1	10			
	0,4	20					
	0,4	10					

Figure 42 : Table de données fournisseurs temporels

Lors de l'exécution de la simulation, notamment de la tâche « Raw materials », Moka va inscrire la date d'exécution de la tâche (03/06/2018 09:01:00) dans la feuille Excel interface. A partir de cette date, un facteur météo s'applique sur la durée calculée : le mois de juin a 90% de chances d'entrainer 2 jours de retard supplémentaires.

Après simulation, on constate sur la Figure 43 que la durée de la tâche « raw material » fut de 277 jours.

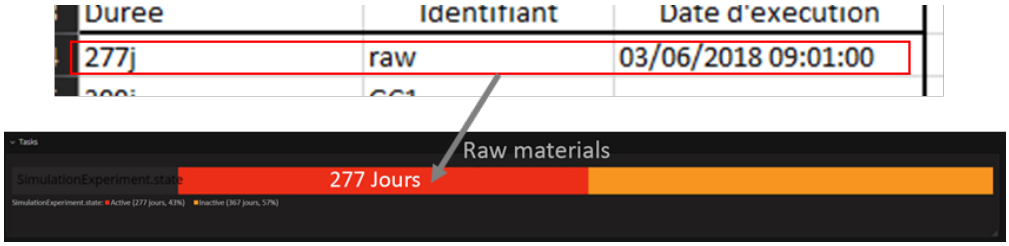


Figure 43 : Résultats simulation Papyrus

9.2.Application générale : Bases de données de gestion des risques

Excel est un outil puissant qui permet de manipuler simplement des données par un utilisateur non informaticien. Son interface graphique et sa popularité en font un outil très utilisé dans le domaine de l'industrie. Il est cependant limité et peu performant quand il s'agit d'interagir avec des outils externes (tel que Papyrus ou autres). L'objectif de cette contribution est de proposer une alternative à l'utilisation d'Excel pour recenser les risques de simulation.
Les systèmes de gestion de bases de données sont des outils conçus pour stocker, délivrer, et faire des opérations sur des grandes quantités de données. C'est pourquoi nous proposons d'interfacer Papyrus avec une base de données locale (SQLite).

Passer par des mécanismes de systèmes de gestion de bases de données offre de nombreux avantages en termes de fonctionnalités et d'interopérabilité avec les autres outils.

L'objectif de cette contribution est de proposer une base de données des risques connecté à une simulation Papyrus dans laquelle l'ensemble des risques vont être stockés dans une table accessible durant la simulation. Passer par une technologie de bases de données permet d'utiliser un standard de stockage très largement utilisé, open source, et facilement accessible par n'importe quel outil afin de générer des scénarios de risques.

Lorsque Papyrus exécute une tâche, elle sera associée à une durée d'exécution native : le temps de traitement nominal d'une machine décrite par le système de production sous Papyrus sans prise en compte des risques. Cette durée peut être prolongée ou réduite par la ligne qui lui est associée dans la base de données de gestion des risques. Cette valeur associée est déterminée par des ingénieurs afin de décrire les probabilités d'occurrence d'un problème ciblé sous la forme d'une équation. La configuration des risques est effectuée à l'aide d'une base de données SQL (voir Figure 44) qui permet aux utilisateurs de stocker les lois de probabilité et les impacts des risques qui sont exécutés pendant la simulation.

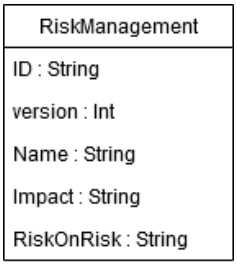


Figure 44 : UML base de données gestion des risques

Dans la Figure 44, le champ « ID » doit correspondre à l'identifiant unique déterminé par le profil UML appliqué sur les tâches dont on veut définir un comportement à risque. Le champ « version » permet de gérer plusieurs risques affectés à une même tâche. Ce champ s'auto incrémente pour chaque nouvelle ligne pour un même ID. Le champ « Name » est un champ libre pour que l'utilisateur puisse décrire la source du danger. Le champ « Impact » contient une équation déterminée par les ingénieurs définissant l'impact qu'aura le risque sur la simulation. Enfin, le champ « RiskOnRisk » contient une chaine de caractère au format JSON contenant l'ensemble des effets qu'aura le risque sur les autres risques de la simulation.

L'équation (1), déterminée par un expert, est un exemple illustrant dans la simulation les délais de fabrication du Génie Civil 1 (GC1). Cette équation est donc décrite pas le champ « Impact » de la Figure 44.

$$t = t1 \times f(x);\ f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad (1)$$

L'équation (1) représente une loi de dégradation temporelle que l'on applique à la tâche GC1. Dans l'équation (1), GC1 possède une durée de fonctionnement native représentée par t1. Cette durée est affectée par une loi de distribution normale f(x). Dans cet exemple, le temps d'exécution de la tâche GC1 sera modifiée au moment de l'exécution du modèle par Moka en fonction de différents paramètres :

- x : un nombre aléatoire entre -1 et 1 généré à l'initialisation du moteur. Comme pour la contribution Excel, ce nombre est généré avec un système de seed, ce qui permet à l'utilisateur de reproduire la simulation avec les mêmes générations aléatoires.
- σ : (peut aussi être appelé GC1.sd pour « standard déviation ») représente l'écart type du risque. Cette valeur n'est pas fixe car elle peut être modifiée à tout moment de la simulation via le champ « RiskOnRisk » de la table RiskManagement de la Figure 44.
- μ : (peut aussi être appelé GC1.m pour « mean ») représente la moyenne de la loi normale. Également définie dans la base de données, cette valeur peut être modifiée via n'importe quel autre risque.

Dans notre exemple, le temps de simulation de GC1 sera modifié par la loi normale décrite par l'équation (1). Cette équation peut cependant être amenée à évoluer au cours de la simulation. En effet, le champ « RiskOnRisk » permet de décrire des événements secondaires qui impacteraient d'autres tâches. La chaine de caractères JSON du Tableau 3 décrit les interactions entre les tâches existantes,

et permet d'introduire de nouveaux risques pendant l'exécution de la simulation. Par exemple, un événement peut impacter les probabilités de causer un problème sur une tâche connexe, ou même ajouter un nouveau type de risque.

```
"riskOnRisk":
[
	{
		"condition":"GC1>0.05",
		"impacts":
		[
			{"target":"GC2.2", "impact":"modify Normal(0.07, 0)"},
			{"target":"GC3", "impact":"add Normal (0.03, 0.5"}
		]
	}
]
```

Tableau 3 : Création de nouveaux risques

Le Tableau 3 décrit le cas d'une possibilité de propagation de panne à partir de la tâche GC1. Le champ « condition » indique la condition à valider pour être dans un cas de propagation de risques. Dans notre cas : le résultat de l'équation (1) doit être strictement supérieur à 0.05. Dans ce cas, deux impacts auront lieux sur le système :

- La modification du risque relatif à la tâche GC2 : l'impact du risque affecté à la tâche GC2, version 2 sera remplacé par une loi normale dont l'écart type sera 0.07 et la moyenne 0. En cas d'absence de précision de la version ciblée (« .2 »), le système modifiera le dernier risque ajouté à la base de données.
- L'ajout d'un nouveau risque relatif à la tâche GC3 : une nouvelle ligne dans la base de données SQL sera ajoutée pour la tâche GC3. L'impact nous indique la création d'une nouvelle loi normale avec pour écart type 0.03 et pour moyenne 0.5. Si une ligne pour ce risque est déjà présente pour la tâche GC3, le système auto incrémentera le champ « version » de la ligne.

10.Conclusion

Ce chapitre s'est articulé autour de l'externalisation d'une partie du modèle Papyrus. En résultat, la modélisation des risques est désormais extérieure à l'outil. Pour répondre à la question Q1, nous avons dans un premier temps proposé le développement d'une extension de Papyrus afin de permettre un point d'entrée/sortie de l'outil. A partir de là, nous avons pu dans un premier temps étendre sémantiquement le modèle de base pour ajouter la notion d'identifiants uniques à chaque tâche. Dans un second temps, la définition d'une extension du moteur d'exécution nous a permis d'agir sur l'ordonnanceur de Papyrus au cours de la simulation afin d'intégrer des modifications lors de l'exécution de certaines tâches (les tâches soumises aux risques).

Les durées des tâches étant modifiables au cours de la simulation, Moka, le moteur de Papyrus est alors capable d'effectuer des requêtes vers un outil externe de gestion de risques pour délocaliser les calculs de risques. Ces deux ajouts permettent une contribution sur l'aspect de la gestion des données d'entrée d'une

simulation à hauts risques. Deux applications ont été proposées à partir de cette contribution :

- Une application industrielle en lien avec ALSOLENTECH permettant l'identification de risques métiers depuis un fichier Excel.
- Un dérivé de l'application industrielle permettant l'identification de risques depuis une base de données SQLite et permettant une accessibilité accrue aux outils de gestion des risques externes.

Les propositions faites dans ce chapitre permettent ainsi une externalisation de la gestion des risques d'un système industriel dans un modèle et outil de gestion des processus.

Dans l'objectif d'orchestrer la simulation de ce modèle de processus dans le temps, dans le chapitre suivant, nous nous intéresserons au standard de co-simulation FMI au travers de deux contributions :

- son articulation avec le standard HLA
- son intégration dans l'outil de M&S Papyrus

Chapitre 4
Interopérabilité du standard HLA avec le standard FMI pour la co-simulation

Sommaire

1.Introduction

La modélisation et simulation de systèmes complexes requiert simultanément l'expertise de plusieurs métiers. Ces compétences sont, souvent, issues de domaines utilisant des concepts et langages variés. Les technologies de simulation distribuée et de co-simulation permettent de répondre à ces problématiques en assemblant des composants de simulation hétérogènes. L'architecture de haut niveau (HLA) est une des spécifications d'architecture logicielle les plus connues et utilisées. Cependant, d'autres alternatives existent : Functional Mock-up Interface (FMI) définit une interface standard pour coupler des simulations de systèmes, ou des sous-modèles. Simple d'utilisation, c'est un standard utilisé dans l'industrie pour ses avantages de rapidité d'implémentation et de garantie de confidentialité. L'objectif de ce chapitre est de s'intéresser à ce standard et les problématiques d'interopérabilité qu'il soulève. Cela va passer par deux problématiques. Dans un premier temps nous discuterons de sa compatibilité avec le standard HLA et tenterons de créer un pont entre deux implémentations de ces standards. Dans un second temps nous adapterons notre outil de modélisation et simulation Papyrus au standard FMI afin de le rendre compatible avec FMI pour la co-simulation.

2.Combiner les standards HLA et FMI

2.1.Inclure dans une fédération HLA, des composants FMU

Afin d'inclure dans une simulation distribuée HLA des composant FMU, un fédéré doit être l'hébergeur. Comme évoqué dans l'état de l'art, un FMU peut embarquer son propre solveur (dans le cas d'un FMU pour la co-simulation), mais il n'est pas autonome pour autant. Il doit être chargé et piloté depuis un décideur afin de gérer son cycle de vie, sa temporalité, et ses interactions avec son environnement au cours de la simulation. Le décideur sera une interface établissant une connexion et gérant les échanges de données avec le(s) FMU exécutant(s). Dans le cas d'une cohabitation HLA-FMI, un composant FMU chargé ne pouvant communiquer qu'avec son décideur, le fédéré HLA a donc la responsabilité d'orchestrer la simulation à travers deux points importants :

- Suivre et contrôler les échanges de données entre la fédération et le FMU
- Synchroniser le temps du FMU avec les mécanismes de gestion du temps implémentés par HLA [136].

L'intégration d'un composant FMU au sein d'un fédéré HLA apporte différents points intéressants :

- Une fois adapté au fédéré, un composant FMU peut être réutilisé dans un nouvel environnement de simulation HLA sans nécessiter de changement structurel ou comportemental.
- Une plus grande interopérabilité d'utilisation due à sa compatibilité avec les standards HLA et FMI

- Les FMU peuvent être créés et testés indépendamment de la simulation avec l'aide d'outils tiers

La principale différence entre FMI et HLA est que HLA fournit des mécanismes spécifiques pour l'échange de données (publication et souscription), et la gestion du temps (il permet de gérer le temps au travers de la fédération et s'assurer que les fédérés fonctionnent à la même cadence), ce qui permet l'intégration de simulation distribuée de composants hétérogènes. Dans notre cas, seul FMI pour la Co-Simulation (CS) peut être considéré dans cette étude. Dans FMI pour « Model Exchange », le solveur n'est pas inclus dans le FMU, il n'est donc pas intéressant de l'ajouter à une fédération HLA.

Une approche permettant la liaison des deux standards consiste en la création d'un fédéré hébergeant un composant FMU. Le fédéré aura la fonction de passerelle gérant le rythme d'avancée du FMU. Il fera transiter les messages du RTI vers le FMU, et rapatriera les messages du FMU vers le RTI. Un FMU n'étant pas construit sur la base des événements, il ne peut pas déclencher de fonction chez son hôte. Dans la littérature, on trouve différentes méthodes permettant d'obtenir une communication bi-directionnelle (prédiction de modèles ([137]), couplage par points de synchronisation [138]). Dans ce document, les mécanismes d'intéropérabilité implémentés entre Moka / HLA et FMI sont uni-directionnels (Moka/HLA → FMI).

Les variables de FMI pour la « Co-Simulation » ne peuvent correspondre qu'aux types de données de bases HLA car le standard FMI ne prend en charge que les types primitifs suivants : Réel, entier, chaîne de caractères, booléen et énumération, contrairement aux attributs HLA pouvant représenter n'importe quelle structure de données. Dans ce contexte, un environnement de simulation utilisant des types de données complexes ne peut pas être directement pris en charge par le FMU, on devra alors faire une conversion de ces données via l'interface HLA.

La structure du fédéré hébergeant le FMU est donc composée de deux éléments :

- Un composant FMU pour la « co-simulation » contenu dans le fédéré
- La couche d'implémentation HLA permettant la communication à la fois avec le RTI et le FMU.

2.2.Application générale

Afin d'expérimenter l'applicabilité de cette proposition, nous avons mis en place un exemple simple capable de démontrer l'utilisation et la viabilité d'un environnement HLA + FMI.
L'objectif de cette section est de concevoir la « preuve de concept » d'un FMU simulé dans un environnement distribué HLA. La fédération est basée sur une simulation de balle rebondissante, beaucoup utilisée dans les domaines de modélisation et simulation. Il s'agit d'une simulation de balle lâchée d'une certaine hauteur et rebondissant sur un sol tout en fournissant des informations sur la hauteur et la vélocité de la balle. Le fichier FMU de la balle a été généré en utilisant

le FMU SDK de Otrnic [139]. Dans ce scénario, deux fédérés vont rejoindre une fédération (voir Figure 45) :

- *Fed A* : fédéré servant de GUI. Il est abonné aux interactions émises par le fédéré hybride. A la réception d'informations, il va tracer la trajectoire de la balle et écrire les données dans un fichier Excel.
- *Hybrid Fed A* : fédéré responsable de la simulation de la balle au travers d'un FMU. Il va faire avancer le temps du FMU au même rythme que son propre temps. A chaque pas de temps, il récupère les données du FMI et les envoient par une interaction.

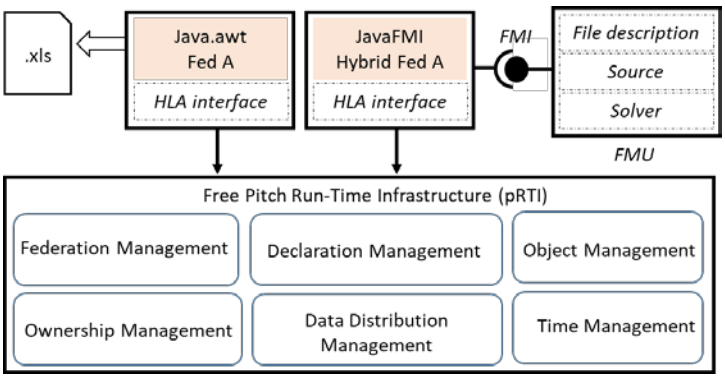


Figure 45 : HLA/FMI fédération

Comme nous pouvons le voir sur la Figure 45, nous utilisons le pRTI Pitch qui nous permet de créer et de gérer une fédération. Après l'exécution de la simulation distribuée, nous pouvons observer le résultat généré par le fédéré A à gauche de la Figure 45. L'échange de données entre les deux fédérés se fait via la déclaration de FOM file sur la Figure 46.

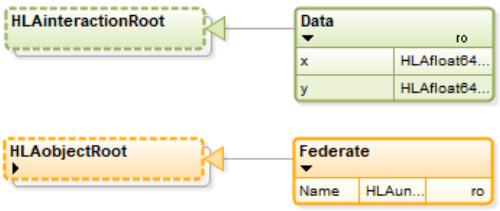


Figure 46 : Fédération Object Model

Les informations décrites sur la Figure 46 sont contenues dans le fichier XML FOM de la Figure 47. La norme HLA prévoit deux types d'échanges de données :

- Les *Interaction Class* ne sont pas persistantes dans le temps et peuvent avoir des paramètres. Elles sont la plupart du temps utilisées pour décrire des entités éphémères pendant une simulation. Les interactions sont utilisées ici pour contenir les coordonnées *X* et *Y* de la balle générée par le FMU et transmises à la fédération via le fédéré hybride.
- *Object Class* sont persistant pendant la simulation. Elles peuvent avoir des attributs pouvant être mis à jour par un fédéré. Dans notre application exemple, *name* sera utilisé pour identifier les fédéré « Fed A » (fédéré de réception des données) et le fédéré *Hybrid Fed A* qui sera le FMU.

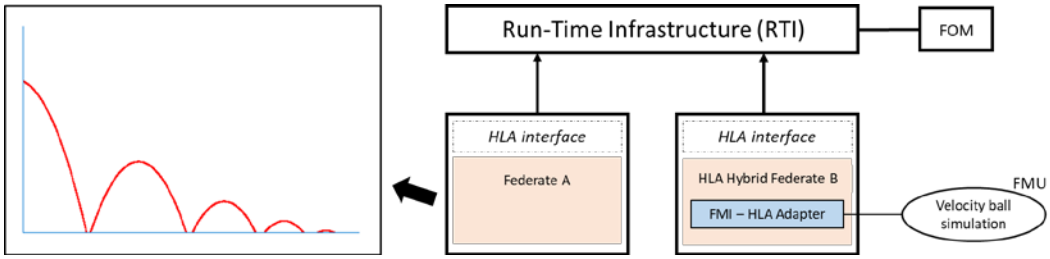


Figure 47 : Fédération hybride HLA

On peut voir sur la Figure 47 l'architecture technique de la fédération hybride. Le fédéré A reçoit des données du RTI selon une interaction souscrite au fédéré hybride (*Hybrid Fed A*). Cette interaction est décrite dans le fichier FOM (*Federation Object Model*) Figure 46 qui contient la structure logique des échanges entre les simulations. A la réception de ces informations, la fédération *A* trace un point sur un graphique (via le package java.awt). Tout au long de la simulation, les données sont calculées par le FMU, envoyées au travers des mécanismes de communication HLA (via les interactions) au second fédéré. Une fois reçues, les données sont tracées sur le graphique, et enregistrées dans un fichier Excel.

On manipule donc dans cet exemple un FMU *bouncing ball* via un ensemble de variables d'entrée/sortie décrites par la Figure 48.

Name	Start	Unit	Plot	Description
® der(h)			☑	velocity of ball
® der(v)			☑	acceleration of ball
® e	0.7		☑	dimensionless parameter
® g	9.81		☑	acceleration of gravity
® h	1		☑	height, used as state
® v			☐	velocity of ball, used as state

Figure 48 : Variables bouncingball

On observe six différentes variables avec lesquelles interagir lors de l'exécution du composant FMU ainsi qu'au cours de sa simulation. Dans notre cas, nous définissons l'accélération de la gravité à 9.81 et la hauteur de départ de 1.

La définition de ces paramètres se fait via un appel à la fonction *writeVariable(String, Object)* qui prend en premier paramètre le nom de la variable décrite par le XML, puis un objet contenant la valeur à injecter.

La lecture d'un paramètre se fait via la fonction *readVariable(String)* prenant en entrée le nom désignant la variable que l'on souhaite lire. La fonction nous retournera un objet que l'utilisateur devra convertir en un type de donnée adéquat (on rappelle que le standard FMI ne gère que les types de données simples).

Une simulation FMU nécessite une configuration temporelle. En effet, avant son exécution, l'utilisateur doit préciser son temps local de départ, et son temps local de fin via la fonction *init(int startTime, int stopTime)*.

Enfin, la synchronisation temporelle des deux simulations est une étape importante. Du point de vue du standard FMI, faire avancer une simulation dans le temps se fait via l'appel de la méthode *doStep(double stepSize)* qui prend en paramètre le temps d'avancement de la simulation. Cette méthode fait avancer l'exécution de la simulation d'une durée déterminée (définie par défaut dans le code), les variables de sorties sont mises à jour en conséquence. Du point de vue de HLA, le temps est géré de façon évènementielle (Event-based time advancement service). Le fédéré hybrid HLA/FMI est défini comme moteur (time regulating) via la fonction « *EnableTimeRegulation()* ». C'est lui qui fait avancer la cadence de la simulation d'événements en événements par l'appel de la fonction « *nextEventRequest()* ». Il est à noter qu'un fédéré peut à la fois être moteur (time regulating) et suiveur (time constrained). De son coté, le fédéré A est défini comme suiveur. L'appel d'avance dans le temps par le fédéré B déclenchera le *timeAdvanceGrant* du fédéré suiveur, et fera avancer son temps local. La Figure 49.A représente l'algorithme de synchronisation entre le FMU et le fédéré hybrid A. La Figure 49.B représente l'algorithme du fédéré A récupérant les données envoyées par l'algorithme A.

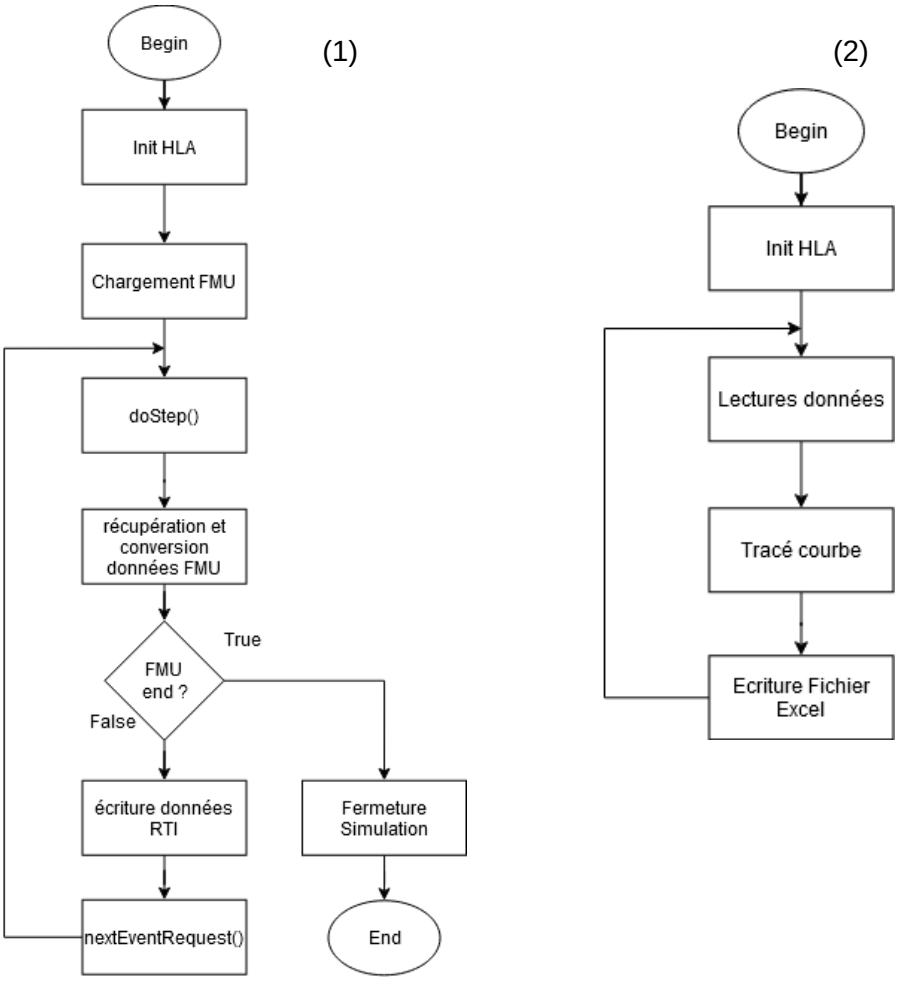


Figure 49 : Algorithme synchronisation FMU/Fédéré

La Figure 49 illustre :

- Les échanges de données entre fédéré hybrid et le FMU ;
- La synchronisation de l'horloge FMI (via la méthode *doStep()*) et l'horloge HLA (via la méthode de requête d'avancement dans le temps *nextEventRequest()*) ;
- La communication entre les deux fédérés HLA.

Cette application a pour rôle de démontrer la faisabilité de l'utilisation d'un FMU au sein d'une fédération HLA. Nous utiliserons la notion de fédéré hybride dans le chapitre final.

2.3.Application industrielle

Dans le cadre industriel, l'utilisation du standard FMI a pour objectif de démontrer la faisabilité d'importer un FMU dans l'outil Papyrus afin de délocaliser des algorithmes, et rendre l'outil ouvert à des standards permettant l'utilisation d'outils connexes. Pour ce faire, la solution du standard FMI fut retenue pour sa garantie de confidentialité des modèles et des simulations importées.

Dans notre cas d'utilisation, il s'agira d'importer un outil de dégradation de performances météorologiques qui, en fonction des données entrée, déterminera un coefficient d'efficacité du système. Selon les données de sortie du FMU, la simulation verra ses performances dégradées ou non. Le cas abordé dans ce chapitre est un exemple commandé par l'entreprise afin de démontrer la faisabilité d'une telle solution.

2.3.1.Modèle d'évaluation de performances

Le cahier des charges commandé par ALSOLENTECH est de permettre l'interaction d'un outil externe, présent dans l'environnement de Papyrus via un standard pivot. Nous explorons ici la norme FMI qui offre des avantages de simplicité d'utilisation au prix de fonctionnalités moindres. L'objectif de cette contribution est de permettre l'interaction (sous la forme d'une dégradation temporelle) d'un modèle « principal » simulé dans l'environnement Papyrus avec un composant externe connecté à la simulation via le standard FMI pour co-simulation.
Pour ce faire, nous allons partir d'un modèle macro de simulation de centrale solaire (voir Figure 50) et ajouter un module externe de gestion des performances lié à son tour à des rapports météorologiques fournis en temps réel, donc lui-même connecté à un service de données météorologiques en ligne : *OpenWeatherMap* (Figure 51).

La Figure 50 représente un modèle de mise en place du projet SoLR² (déjà explicitée dans le chapitre 3). Certaines étapes de ce modèle nécessitent l'intervention d'un module externe de calcul de performances liées aux conditions météorologiques. Leurs délais d'exécution seront déterminés en fonction des résultats de l'outil. La première tâche affectée par ce dernier est *Mobile Factory Carrying* qui dépend de la pluviométrie sur place. Une fois l'usine déployée, une

batterie de test est effectuée pour déterminer la puissance effective de la centrale (tâche *Test Mobile Factory Production* sur la Figure 50). L'efficacité de ces tests va également être déterminée avec l'aide du module externe puisqu'elle est directement en lien avec le taux d'ensoleillement. Enfin, s'enchainent les phases finales de l'installation de la centrale, elles aussi dépendantes des conditions météorologiques.

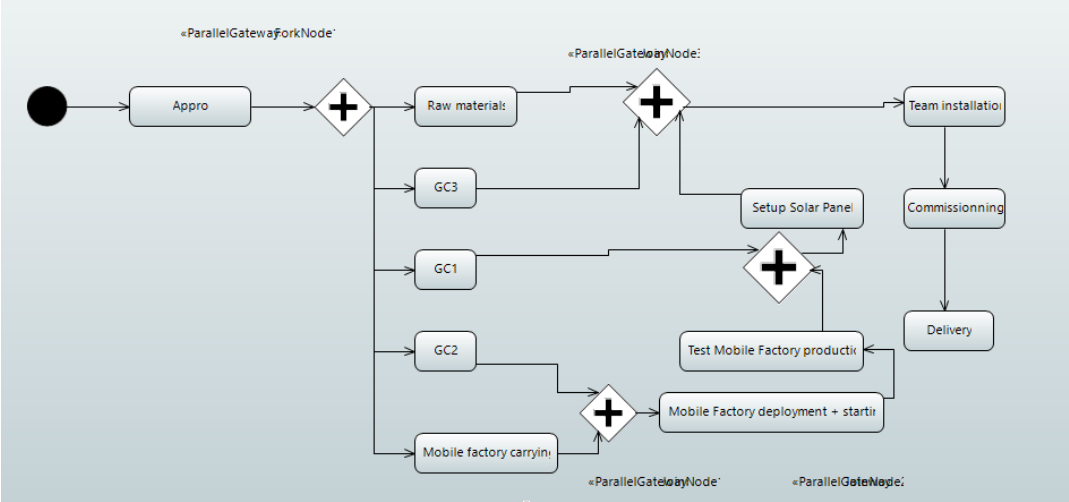


Figure 50 : Simulation Papyrus

Nous allons donc créer une seconde extension du moteur d'exécution Moka permettant des interactions entre la simulation et un composant FMU (la première extension ayant été présentée dans le chapitre 3). On observe donc l'architecture suivante (Figure 51).

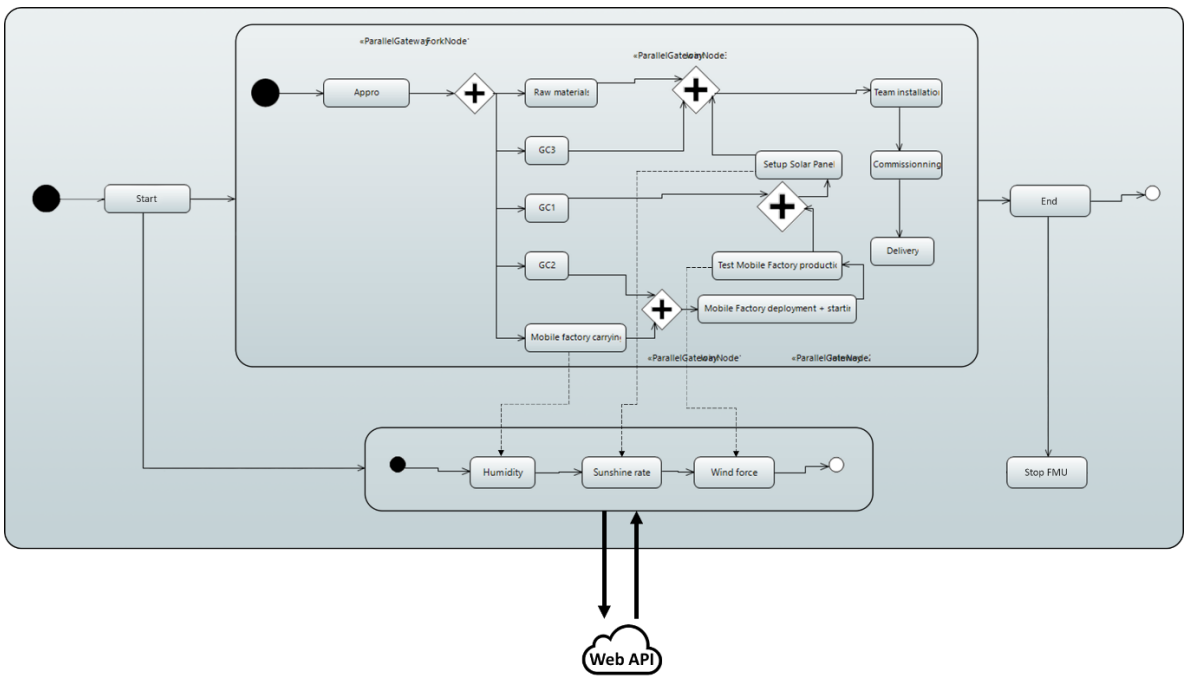


Figure 51 : Articulation Papyrus & FMU

La Figure 51 représente les différents composants et standards utilisés pour mettre en place cette démonstration et illustre les différents points que nous aborderons pour démontrer notre approche. Le standard FMI va permettre à Papyrus de communiquer avec un modèle exécuté parallèlement au sein du FMU. Ce modèle, plus simple, va déterminer des dégradations temporelles de certaines tâches de la simulation principale.
Dans la suite de ce chapitre, on considère qu'un fichier de configuration est associé à la simulation. Le fichier est récupéré par Papyrus au démarrage de la simulation et permet de stocker des informations de base comme la localisation du site de production, etc.

2.3.2.API Weather

Le module développé pour démontrer l'utilisation du standard FMI au sein de la simulation est un outil d'évaluation de performances liées aux conditions météorologiques. *OpenWeatherMap* propose à sa communauté l'accès à une API permettant des requêtes sur les conditions météorologiques d'un lieu à une date précise sur les 40 dernières années.

L'API peut prendre en entrée les paramètres résumés sur le Tableau 4 :

Lat	Latitude
Lon	Longitude
Weather	Localisation par mot clef
Type	Type de requête
Start	Date de départ du relevé météo
End	Date de fin du relevé météo
Cnt	Quantité de data retourné voulue
apiID	Clef d'identification de l'API

Tableau 4 : Paramètres d'entrée API

L'appel se fait via la méthode *openConnexion()* de la librairie standard *java.net.URLConnection visible sur la* Figure 52.

```
try {
        URL myURL = new
        URL("http://api.openweathermap.org/data/2.5/weather?q="+location +
  "&APPID="+key);
        URLConnection yc = myURL.openConnection();
        BufferedReader in = new BufferedReader(
        new InputStreamReader(yc.getInputStream()));
        String inputLine;

        while ((inputLine = in.readLine()) != null)
                result += inputLine;
        in.close();
} catch (Exception e)
{
        e.printStackTrace();
}
```

Figure 52 : Récupération des données de l'API

Les résultats sont reçus et stockés dans la variable *result* du code Figure 53 sous la forme d'une chaine JSON :

```
{
        “message” : Paramètres internes,
        “cod” : Paramètres internes,
        “city_id” : Identifiant de la zone,
        “calctime” : Paramètres internes;
        “list” :
        [{
                “dt” : date du relevé météo
                “main” : {
                        “main.temp” : température en Kelvins
                        “main.feels_like” : température ressenti en Kelvins,
                        “main.pressure” : Pression atmosphérique en hPa,
                        “main.humidity” : Humidité dans l’air en %,
                        “main.temp_min” : Température minimum atteinte
                        “main.temp_max” : Température maximum atteinte
                        “main.sea_level” : Pression atmosphérique du niveau de mer hPa
                        “main.grnd_level” : Pression atmosphérique niveau du sol hPa
                        }
                “wind” : {
                        “wind.speed” : Vitesse du vent m/s
                        “wind.deg” : Direction du vent en degrés
                        }
                “clouds” : {
                        “clouds.all” : % de présence nuages
                        }
                “rain” : {
                        “rain.1h” : Volume d’eau tombé durant l’heure précédente
                        “rain.3h” : Volume d’eau tombé Durant les 3 dernières heures
                        }
                “snow” : {
                        “snow.1h” : Volume de neige tombé durant l’heure précédente
                        “snow.3h” : Volume de neige tombé durant les 3 dernières heures
                        }
                “weather” : {
                        “weather.id” : ID des conditions météorologiques
                        “weather.main” : Groupe de paramètres internes
                        “weather.description” : Description des conditions météo
                        “weather.icon” : Identifiant d’un icône correspondant
                }
        }]
}
```

Figure 53 : JSON de résultat de l'API

Les données peuvent être ainsi récoltées et traitées par le composant FMU. En fonction des données d’entrées fournies par la simulation Papyrus, l’outil effectuera des requêtes à l’API et déterminera des valeurs de dégradations à renvoyer au modèle principal.

2.3.3.Wrapper un FMU

Le module FMU que nous avons développé pour démontrer les interactions possibles entre Papyrus et le standard FMI est conçu à partir d’un programme java. Ce module a pour objectif de compléter le modèle principal simulé pour cette démonstration (voir Figure 50). Sa simulation consiste en 3 phases, pour les 3 interactions avec Papyrus visibles dans la Figure 51 et la Figure 54.

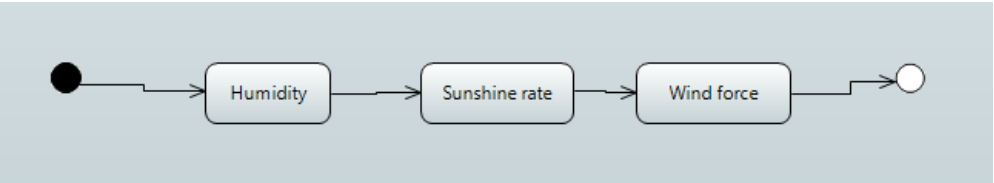


Figure 54 : Etapes d'exécution du FMU

Chacune des 3 tâches qui composent le FMU vont effectuer une requête et un calcul différent. La première tâche effectue une recherche sur l'API web de l'humidité moyenne hebdomadaire sur les 10 dernières années. Par exemple, si nous sommes le 1er janvier 2020 à l'exécution de la tâche 1, nous déterminerons une moyenne d'humidité des 10 dernières semaines 1. A partir de cette valeur obtenue en pourcentage, nous dégraderons le temps d'exécution de la tâche de livraison de l'usine mobile du modèle Papyrus. Les étapes 2 et 3 sont associées à d'autres tâches du modèle et permettent d'interroger l'API météo sur les taux d'ensoleillement et sur la force du vent.

La seconde tâche du composant a pour objectif de collecter des informations statistiques relatives à la présence de nuages durant les 10 dernières années. La troisième en rapport avec la force du vent.

Une fois les algorithmes de calcul et les interactions avec l'API météo fonctionnels, nous convertissons le programme java en un composant compatible avec le standard FMI. Pour cela, nous utilisons un ensemble de fonctions fournies par la bibliothèque javaFMI permettant d'importer un composant FMU pour la co-simulation au sein d'une application java. La bibliothèque permet également d'exporter une application java en FMU compatible avec le standard.

Le standard nous indique donc la procédure à suivre pour rendre une application compatible FMI. La classe principale doit hériter de FmiSimulation et doit redéclarer des fonctions de cette dernière :

- *Define()* : (Figure 55) est exécuté au chargement d'un fichier FMU. Il décrit le modèle chargé par le décideur, notamment les variables d'entrée et sortie, leurs types et leurs variabilités dans le temps.

```
@Override
 public Model define() {
 return model(ModelName).canGetAndSetFMUstate(true)
 .add(variable(Location).asString().causality(input).variability(discrete))
 .add(variable(ApiKey).asString().causality(input).variability(discrete))
 .add(variable(MainWeather).asString().causality(output).variability(discrete))
 .add(variable(WeatherDescription).asString().causality(output)
  .variability(discrete))
 .add(variable(Temperature).asReal().causality(output).variability(discrete))
 .add(variable(Pressure).asReal().causality(output).variability(discrete))
 .add(variable(Humidity).asReal().causality(output).variability(discrete))
 .add(variable(TempMin).asReal().causality(output).variability(discrete))
 .add(variable(TempMax).asReal().causality(output).variability(discrete))
 .add(variable(JSon).asString().causality(output).variability(discrete));
 }
```

Figure 55 : Description les paramètres d'un FMU

Dans le code de la Figure 55, la première ligne retourne notre modèle que l'on référence via son nom *ModelName*. Ce modèle est défini par différentes fonctions que l'on ajoute en suivant. La fonction *canGetAndSetFMUstate(bool)* détermine si on autorise une sauvegarde de la simulation courante afin de la recharger ultérieurement dans des états prédéfinis (dans notre cas d'utilisation, cette fonction est autorisée). Par la suite, on référence l'ensemble des variables d'entrée/sortie via la fonction *add()*. Cette fonction permet d'indiquer au décideur le nom des

variables (*variable(String)*), et leur type (*asType()*) on rappelle que le standard FMI 2.0 ne prend en charge que les types primitifs.

- *Init()* : (Figure 56) est appelé après le chargement du FMU. Il permet d'initialiser les variables d'entrée, sortie, et internes du composant.

```
@Override
 public Status init() {
 registerString(Location, () -> location, value -> location = value);
 registerString(ApiKey, () -> apiKey, value -> apiKey = value);
 registerString(MainWeather, () -> mainWeather);
 registerString(WeatherDescription, () -> weatherDescription, value ->
weatherDescription = value);
 registerReal(Temperature, () -> temp, value -> temp = value);
 registerReal(Pressure, () -> press, value -> press = value);
 registerReal(Humidity, () -> humid, value -> humid = value);
 registerReal(TempMax, () -> temp_max, value -> temp_max = value);
 registerReal(TempMin, () -> temp_min, value -> temp_min = value);
 registerString(JSon, () -> json, value -> json = value);
 return Status.OK;
 }
```

Figure 56 : Initialisation des paramètres d'un FMU

A l'exécution du FMU, on associe les variables du programme aux entrées/sorties de la simulation en passant par des mots clefs (dans la première ligne, on associe la variable location au mot clef *Location*). On initialise également sa valeur par défaut.

- *doStep(double)* : (Figure 57) est la fonction utilisée par le décideur pour faire avancer le temps de la simulation. On passe ici en paramètre le temps que l'on souhaite faire s'écouler pour changer l'état des variables internes de la simulation.

```
@Override
public Status doStep(double stepSize) {
 executionSteps = executionSteps + stepSize;
 recalculate();
 return Status.OK;
}
```

Figure 57 : Avance dans le temps d'un FMU

Dans le code de la Figure 57, on incrémente l'horloge locale du programme et on lance une fonction qui recalcule l'état des variables de sortie en fonction de cet avancement dans le temps. En d'autres termes, lors de l'exécution de la fonction *doStep()*, on passe à l'étape suivante de la Figure 54.

- *getState(String)* / *setState(String)* : sont des fonctions permettant de sauvegarder l'état général de la simulation (l'ensemble des variables qui la composent) afin de pouvoir rejouer des scénarios.
- *reset()* : est une fonction prévue par le standard qu'un utilisateur peut appeler pour remettre à zéro les variables de la simulation.

- *terminate()* : met fin à la simulation.

Une fois les différentes règles du standard respectées, on est en mesure de compiler les fichiers en un fichier .jar, puis en un .fmu avec l'aide du fmu-builder via la commande Figure 58.

```
java -jar fmu-builder-<version>.jar user.jar [-n fmu-file-name] [l path/to/dependency.jar …] [-p plateform]
```

Figure 58 : Compilation d'un FMU

La commande visible sur la Figure 58 permet donc de générer un fichier compilé .jar en un .fmu. Le fichier est alors prêt à être importé dans l'outil Papyrus.

2.3.4.Intégration à Papyrus

L'ensemble des interactions entre Papyrus et le standard FMI se font via une extension du moteur Moka personnalisée. L'objectif ici est de reprendre les concepts du fonctionnement de Papyrus énoncés dans le chapitre 3, et de développer un second module Papyrus. Cet ajout fonctionne de manière incrémentale, c'est à dire qu'il permet d'apporter de nouvelles fonctionnalités, tout en conservant les propriétés rendues disponibles par les autres modules implémentés.
Pour assurer la communication avec un module FMU, Papyrus doit se comporter comme un décideur vis-à-vis du composant simulé (qui sera alors un exécutant). Il est donc nécessaire de charger l'instance de notre simulation avant le début de l'exécution de la simulation, lors des phases d'initialisation du moteur Moka. Les interactions avec le FMU se feront via des Advices ajoutés aux Visiteurs des tâches nécessitant une communication avec le composant déporté.

La création d'une instance de FMU se fait via la librairie JavaFMI avec l'appel de la classe Simulation. Il est nécessaire de faire cette instanciation avant l'exécution de la simulation, dans la méthode *doPresRunAction()* (visible Figure 59) du moteur. On rappelle que cette fonction est exécutée par le moteur une fois son initialisation terminée, juste avant le démarrage de la simulation Papyrus. L'instance du FMU est donc en mémoire de l'objet qui contient le moteur. Chaque Advice pourra avoir accès à cette zone mémoire et communiquer avec le composant déporté.

```
void doPresRunAction() {
        […]
        Simulation simulation = new Simulation(filePath);
        //initialisation FMU
        simulation.init(0,10);
        simulation.write("ApiKey").with("9dc74dc8cacaf9a121a0e92b914b0a68");
        simulation.write("Location").with(location);
        […]
}
```

Figure 59 : Fonction doPresRunAction() d'une extension Moka

La gestion de temps entre le moteur d'exécution Moka et le composant FMU se fait de façon séquentielle. A chaque exécution d'une tâche Papyrus faisant référence au composant FMU, le moteur Moka avance automatiquement le temps du FMU d'un pas de temps défini par l'utilisateur (pas de temps inscript dans le fichier de configuration de la simulation). Ainsi, chaque intéraction avec un FMU le fait automatiquement avancer d'un pas de temps, jusqu'à son état final. La Figure 60 représente la gestion du temps d'un FMU par le moteur Moka.

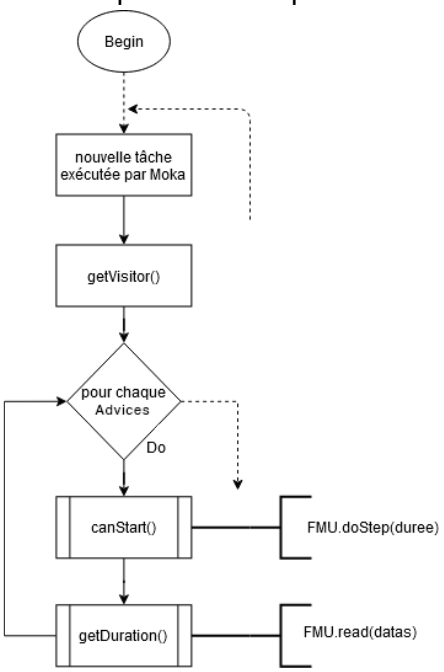


Figure 60 : Algorithme synchronisation Moka / FMI

Lors de l'exécution d'une tâche, Moka interroge chaque visiteur (voir boucle de la Figure 60). Pour chaque advice faisant référence à un composant FMU, la méthode *canStart()* avance le temps du FMU ciblé d'une valeur définie par l'utilisateur. Dans un second temps, la méthode *getDuration()* interroge le résultat du modèle FMU, pour appliquer son résultat à la simulation Papyrus. Le résultat doit être une durée interprétable par le moteur Moka.
Enfin, la fonction *doPostRunAction* (la Figure 61 est exécutée en fin de simulation) permettra de libérer le FMU de la mémoire du moteur.

```
Void doPostRunAction() {
       simulation.terminate();
}
```

Figure 61 : Fonction doPostRunAction() d'une extension Moka

L'Advice personalisé référence une instance du moteur (*EngineExample* sur Figure 62) afin de récupérer l'instance du FMU et communiquer avec ce dernier.

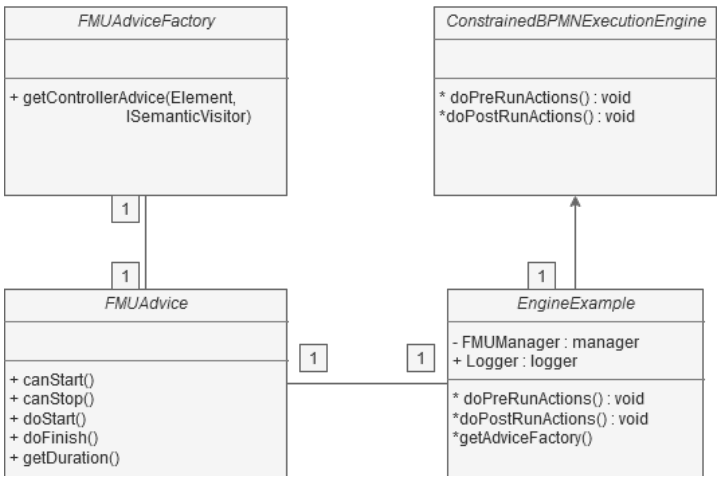


Figure 62 : Architecture extension Moka pour FMU

La Figure 62 représente l'architecture logicielle de cette proposition. La méthode *getAdviceFactory()* de l'*EngineExample* permet de récupérer la liste des Advices associés à chaque Visiteur. On ajoute donc un Advice personalisé (*FMUAdvice*) implémentant les méthodes standard (*canStart(), canStop(), doStart(), doFinish(), getDuration()*).

Depuis l'interface graphique de Papyrus, l'utilisateur doit ajouter le nouveau profil FMU aux tâches nécessitant un échange avec notre composant météorologique. Au lancement d'une simulation, le FMU est chargé en mémoire puis configuré. La simulation débute et MOKA analyse le modèle pour son exécution. Lorsqu'il arrive sur la première tâche de notre modèle, il fait appel à la première étape de notre FMU. La méthode *canStart()* de l'Advice FMU renvoie vraie, et la méthode *getDuration()* est appelée. A cet instant, on avance le temps du FMU avec la fonction *doStep()*, puis on lit la sortie du FMU en appelant son instance via le moteur et en utilisant la méthode *read(nom de la variable)*. Le FMU délivre au programme le pourcentage l'humidité moyenne hebdomadaire sur les 10 dernières années. A partir de cette valeur, la durée de la tâche sera recalculée et dégradée dans le but de simuler des contraintes météorologiques. Après exécution du modèle sans le module de gestion des risques météorologiques connecté, on obtient les résultats de simulation visibles sur la Figure 63.

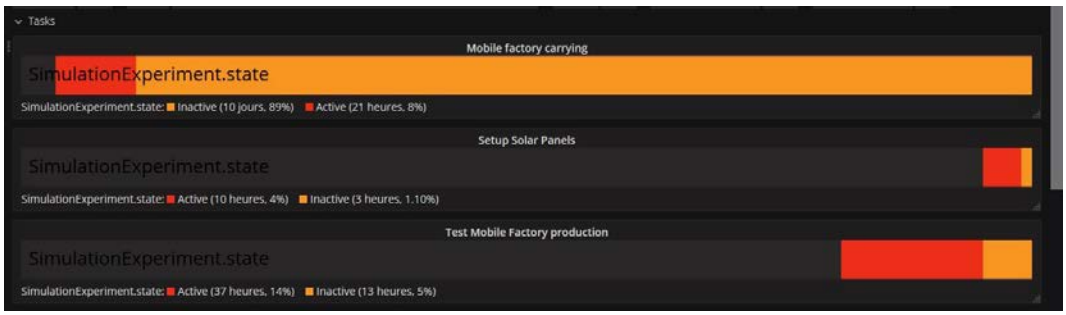


Figure 63 : Résultat de simulation Papyrus sans FMU de gestion des risques

Après simulation sans gestion des risques météo, la première tâche *Mobile factory Carrying* met 21 heures à s'exécuter. La tâche *Test Mobile factory* production met 37 heures, et Setup *solar panels* 10 heures. Dans un second temps, nous

connectons le FMU de gestion des risques météo. Les résultats de cette simulation sont visibles sur la figure Figure 64.

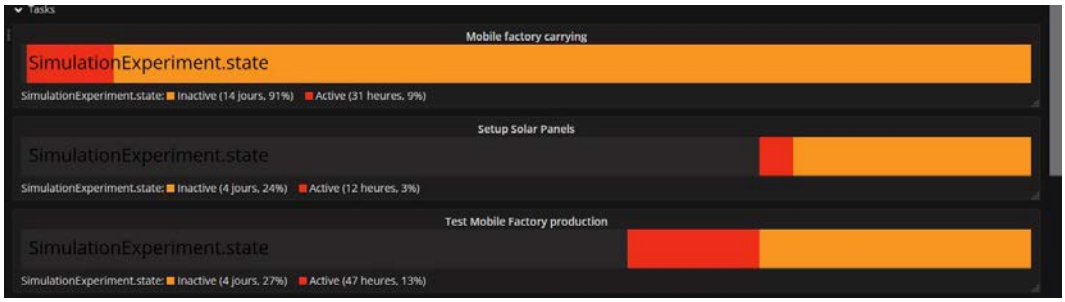


Figure 64 : Résultat de simulation Papyrus avec FMU de gestion des risques

La simulation avec le module de gestion des risques météo impacte la durée des tâches de façon significative. Le Tableau 5 illustre l'impact du module sur le fonctionnement nominal de la simulation.

Tâche	Durée initiale	Durée avec risque	Impact du risque
Mobile factory carrying	21 heures	31 heures	+10 heures
Test mobile factory production	37 heures	47 heures	+10 heures
Setup solar panels	10 heures	12 heures	+2 heures

Tableau 5 : Comparaison des résultats avec et sans gestion des risques météo

3.Conclusion

Les contributions apportées dans ce chapitre concernent le standard Functional Mock-up Interface (FMI) pour la co-simulation. Dans un premier temps, nous avons proposé de créer un pont permettant des échanges entre les standards FMI et HLA. L'objectif était de démontrer la faisabilité d'exécuter un composant FMU au sein d'une fédération HLA. Nous avons démontré que des synchronisations et des échanges de données étaient possibles en tenant compte des limitations de chacun des deux standards. Cette contribution répond à la première problématique de ce chapitre Q2.1 : *Comment créer un pont entre les deux standards de co-simulation HLA et FMI afin d'exécuter un FMU au sein d'une simulation distribuée HLA*.

Dans un second temps, nous avons implémenté une extension à Papyrus permettant son interopérabilité avec le standard FMI (pour la co-simulation). Cette contribution répond à la seconde problématique de ce chapitre Q2.2 : *Comment charger et exploiter un FMU pour la co-simulation au sein de l'environnement Papyrus*. Cet ajout permet à l'outil de charger et interagir avec un composant FMU au cours de la simulation (prise en charge de l'avancement du temps et échange de données). Cette fonctionnalité qui faisait partie du cahier des charges d'ALSOLENTECH permet d'ouvrir l'outil à un standard largement exploité dans le

domaine industriel. Dans le même temps, il permet à Papyrus de communiquer avec des éléments de son environnement comme des objets connectés, des machines externes, serveurs web / API, etc.
Dans le chapitre suivant, nous aborderons le second standard de simulation distribué évoqué dans ce chapitre : HLA.

Chapitre 5
Orchestration d'une simulation distribuée

Sommaire

1.Introduction

Les objectifs de ce chapitre peuvent être réunis en deux questions issues des problématiques déterminées par l'analyse de l'état de l'art :

- Q3.1 : *Comment orchestrer l'exécution d'une simulation distribuée ? Comment un fédéré décideur peut piloter des fédérés exécutants ?*
- Q3.2 : *Comment intégrer un fédéré HLA à la structure de Papyrus de façon qu'il puisse piloter une fédération ?*

Répondre à la première interrogation consiste à définir un ensemble de règles au travers d'un formalisme graphique et de la norme HLA. Décrire de façon simple, pratique et non ambiguë le scénario d'une simulation distribuée est un objectif non résolu dans la littérature que nous souhaitons atteindre.
Le langage BPMN se révèle être un outil intéressant pour modéliser un enchaînement logique de simulations distribuées. Comme nous l'avons vu précédemment, sa lisibilité, son accessibilité et sa flexibilité en font un standard particulièrement utilisé dans le monde de l'entreprise. En effet, il nous permet de rapidement visualiser et comprendre un scénario représenté.
La première proposition de ce chapitre a pour objectif de répondre à la question Q3.1. Pour ce faire, nous proposerons dans un premier temps l'implémentation d'une surcouche au standard HLA permettant d'orchestrer une simulation distribuée en fonction d'un modèle simple, un diagramme BPMN. Cette proposition est divisée en deux sous-points : Le premier consiste en l'implémentation d'un mécanisme de décideur-exécutant au sein d'une fédération HLA. Le second sous-point consiste en l'interprétation d'un modèle BPMN par un fédéré décideur pour orchestrer l'exécution d'une simulation distribuée.
Dans un second temps, et pour répondre à la question Q3.2, nous proposerons l'utilisation des mécanismes d'orchestration HLA (décrits par le premier point) appliqués à l'outil Papyrus.
Durant ce chapitre, nous étudierons dans un premier temps l'approche conceptuelle et technique d'une surcouche HLA permettant l'implémentation d'un mécanisme de décideur/exécutant. Puis, nous verrons deux utilisations de cette proposition, dans le cadre d'un projet collaboratif international. La première est une approche théorique servant de preuve de concept à l'orchestration des fédérés via un composant décideur. Une fois cette approche testée et validée, nous allons l'employer au travers d'un outil de modélisation et de simulation afin de piloter une fédération HLA via l'outil Papyrus.

2.Approche décideur/exécutant dans HLA

La norme HLA est un standard nécessitant une maitrise des connaissances en programmation objet ainsi qu'en modélisation et simulation. Que ce soit pour implémenter une simulation distribuée, ou exécuter une solution clé en main (comme des démonstrations fournies par Pitch), une maîtrise relativement avancée du standard IEEE HLA 1516-2010 est nécessaire. Cette particularité rend le standard difficilement accessible par un grand nombre d'utilisateurs.

Comme on peut le voir sur l'interface de la Figure 65, Pitch, tend à faciliter ce point en proposant une interface simple pour visualiser l'état d'une fédération.

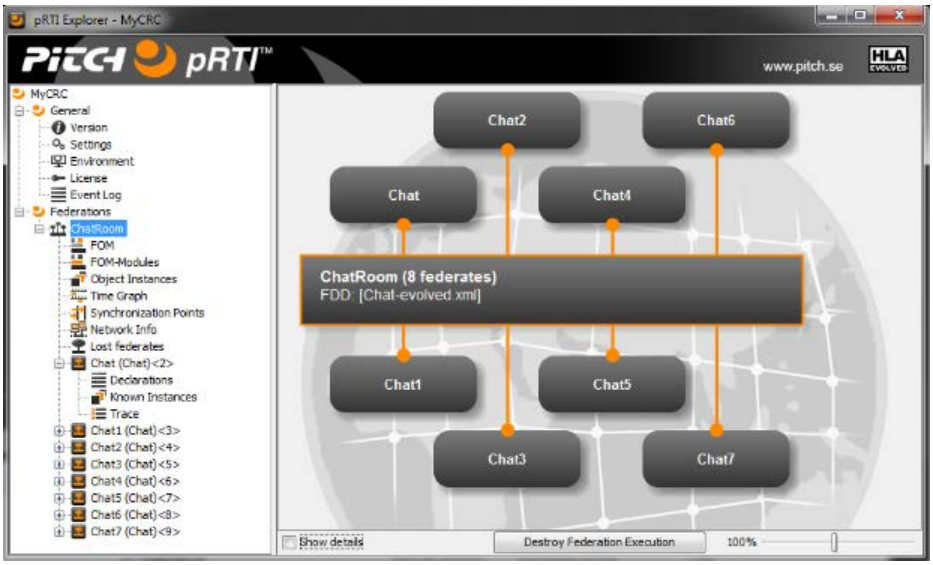


Figure 65 : Interface pitch RTI

Grâce à Pitch, l'utilisateur peut visualiser en temps réel les fédérés connectés à la fédération, les différents fichiers de configuration du RTI, l'écoulement du temps propre à chaque fédéré, etc.
Cependant, il n'existe pas dans le standard HLA d'outil graphique permettant de définir un scénario de simulation. On entend par « scénario de simulation », un moyen de définir simplement un ordre d'exécution des différentes simulations qui composent une fédération HLA. Ce point est un aspect important car il permettrait de définir / redéfinir simplement (grâce à un langage graphique) et rapidement le comportement d'une simulation distribuée.
Pour ce faire, la norme BPMN offre l'avantage de représenter un processus d'entreprise de façon graphique et compréhensible par tout le monde. Après étude des différents outils de modélisation graphique, c'est ce standard que nous allons utiliser pour définir un schéma de simulation distribuée.
Définir un scénario de simulation dans un document unique, revient à centraliser une information décidant de l'ordre d'exécution des différentes simulations distribuées. Le concept « d'ordre d'exécution » introduit la notion de mise en attente d'un fédéré. Il est question ici de lancer l'exécution de fédérés, de les connecter à la fédération, et d'attendre un ordre spécifique pour lancer leurs algorithmes de simulation. Ce schéma doit donc être prioritaire sur tous les fédérés d'une fédération. Une telle chose est impossible dans le standard actuel, tel qu'il est décrit par les spécifications HLA. Chaque fédéré est autonome et ne peut être priorisé que par l'ordre des messages échangés. C'est la raison pour laquelle une orchestration de simulation distribuée HLA n'a encore jamais été proposée.
Afin de mettre en place un système de scénarisation d'une simulation distribuée, il est impératif d'implémenter un mécanisme de décideur-exécutant. Le décideur, ayant pour ressource un diagramme BPMN décrivant l'ordre d'exécution des fédérés doit être en mesure de piloter les fédérés décrits par le schéma. Ici, la notion de pilotage se traduit par un ensemble de règles pré-établies qui vont

permettre une communication entre le décideur et les différents fédérés (les exécutants) par le biais d'intéractions HLA.
D'après le standard HLA, un fédéré décideur, pilotant ses semblables n'est pas réalisable car un fédéré est une simulation autonome, qui n'a besoin que d'un RTI pour fonctionner. Nous proposerons donc dans ce chapitre, le développement d'une surcouche au standard HLA permettant d'implémenter ce mécanisme décideur-exécutant.

2.1.Fédéré exécutant

Comme énoncé précédemment, cette proposition est une surcouche au standard HLA illustrée par la Figure 66, qui va se superposer par-dessus des mécanismes déjà implémentés par les différents frameworks employés. Le fédéré-exécutant est avant tout un fédéré classique, sur lequel on va venir encapsuler la couche de gestion décideur-exécutant, qui pilotera la simulation contenue dans le fédéré.

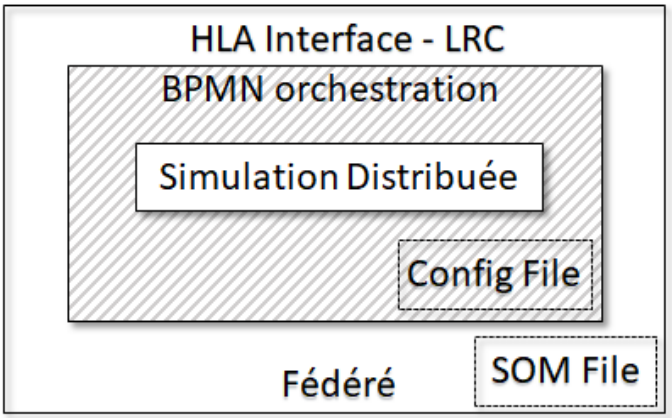


Figure 66 : Vue en couche d'un fédéré « exécutant »

Le schéma Figure 66 représente l'architecture sous forme de "couches" d'un fédéré-exécutant. Comme le modèle OSI du protocole TCP/IP, chaque couche s'appuie sur sa couche inférieure, et sera utilisée par sa couche supérieure.
La communication entre les fédérés (décideur et exécutant) se fait par l'intermédiaire des mécanismes d'échanges déjà présents dans le standard. La couche HLA est donc la plus à l'extérieur et servira de pilier aux étages supérieures. Elle contient l'ensemble des composants nécessaires à un fédéré pour faire partie d'une fédération, à savoir le *LRC* : Local RTI Component, permettant l'interface avec le RTI distant, ainsi que son Fichier *SOM* : Simulation Object Model, décrivant l'ensemble des données à partager avec les autres fédérés.
On trouve dans le fichier SOM l'ensemble des "Objets" et "Interactions" liés à la simulation distribuée en elle-même (ses propres variables locales à partager avec les autres fédérés). On y trouve également un ensemble d'objets et d'interactions nécessaires au bon fonctionnement de la couche supérieure "BPMN Orchestration" permettant la gestion du système décideur-exécutant. Ainsi que les

mécanismes de management du temps. Dans cette application, la régulation du temps HLA sera désactivée.

La couche intermédiaire, "BPMN Orchestration" a deux fonctions :

- Piloter la Simulation Distribuée de la couche supérieure.
- Gérer les communications entre fédéré décideur et fédéré exécutant.

Le contrôle de la simulation distribuée se fait via différents messages qui seront envoyés par le décideur à l'exécutant. En fonction, du contenu du message reçu, le fédéré effectuera une des actions suivantes sur la simulation :

- Démarrage de la Simulation : Message "START"
- Mise en pause de la Simulation : Message "PAUSE"
- Reprise de la Simulation : Message "RESUME"
- Arrêt, destruction de la simulation, destruction du fédéré : Message "KILL"

Au lancement d'un fédéré, après les étapes d'initialisation imposées par le standard HLA, la couche d'orchestration met automatiquement la simulation en pause. Elle est en attente de recevoir le signal "START" du décideur. A la réception de ce message, la simulation est libérée. A l'inverse, lors de la réception d'un signal PAUSE, la couche d'orchestration stope le déroulement de la simulation, ses processus internes ainsi que les communications avec les fédérés extérieurs sont interrompus. De la même façon, la réception d'un signal "RESUME", permet de relancer la simulation et le signal "KILL" de la détruire ainsi que le fédéré dans sa globalité. A l'inverse, un fédéré envoyant le signal "KILL", signifie que sa simulation locale est terminée.

Au sein de cette couche logicielle, les différents messages transitent via les mécanismes de communication fournis par HLA sous la forme d'interactions : Figure 67.

Figure 67 : HLA interaction « Message »

Et d'objets : Figure 58.

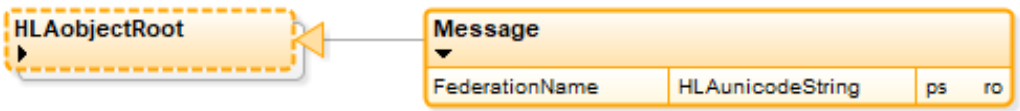


Figure 68 : HLA Object « Message »

L'objet partagé entre tous les fédérés est une chaîne de caractères ("FederationName"). Il permet à chacun des exécutants, à la réception d'un message, de vérifier qu'il est bien connecté à la bonne fédération.
L'interaction "Message" est, de par sa nature, une information non persistante. Elle est créée par un fédéré, et sera détruite au cours du temps. Dans notre cas, cette interaction sera un message adressé à un fédéré particulier. En l'occurrence, la communication est bi-directionnelle. Le décideur peut envoyer un message à un des exécutants, et inversement, un des exécutants peut envoyer un message au décideur.

L'interaction "Message" se compose de plusieurs éléments :

- Id : L'identifiant unique de l'expéditeur du message
- Dst : L'identifiant unique du destinataire
- Msg : Contient l'information à faire transiter (START, PAUSE, RESUME, STOP, KILL)
- Confirm : Est utilisé pour valider un ordre reçu

Pour que la couche BPMN Orchestration fonctionne, l'ensemble des fédérés (décideur et exécutants) ont besoin d'un fichier de configuration commun. Ce fichier texte va contenir trois informations. La première est la liste des fédérés présents dans la simulation, avec pour chacun, leur identifiant unique. Cette information va permettre d'implémenter une communication directionnelle au sein de la fédération. La deuxième information consiste en la désignation du fédéré qui sera décideur. Son identifiant est précisé à l'ensemble des exécutants afin qu'ils exécutent l'ordre reçu par cet identifiant unique. Enfin, la troisième information est une sécurité : le nom de la fédération est écrit en dur dans ce fichier. Ainsi, à la réception d'un message, chaque fédéré vérifie que l'Object *FederationName* est bien le même que celui précisé dans le fichier de configuration.

Afin de garantir la priorité décideur/exécutant, et les communications bi-directionnelles, une procédure d'initialisation visible sur la Figure 69 est nécessaire à chaque démarrage de fédération. En effet, comme nous l'avons décrit plus tôt, la couche BPMN Orchestration fonctionne par-dessus le standard HLA. Il est donc nécessaire d'initialiser une fédération, et ses moyens de communication standards.

Ainsi, lors de la phase d'initialisation d'un fédéré, on peut observer une première phase d'instanciation des différents objets HLA (Init HLA). Juste après, arrive la phase d'initialisation de la couche BPMN Orchestration. Lors de cette phase, un certain nombre de variables et de classes sont instanciées. C'est aussi lors de cette phase que l'on vient lire le fichier de configuration décrivant l'ensemble des participants de la simulation distribuée :

- ID de chaque acteur
- Nom de la fédération
- ID du fédéré exécutant

Une fois cette initialisation effectuée, le fédéré est maintenant exécutant et attend l'ordre d'exécution du décideur. Un signal "START" est en attente. A sa réception,

une série de tests est effectuée afin de vérifier qu'il s'agit bien d'un message provenant du décideur. Le fédéré s'assure que l'expéditeur du message porte bien l'identifiant unique du décideur, et que le nom de la fédération dont il fait partie est bien la même que celle contenue dans le fichier de configuration.

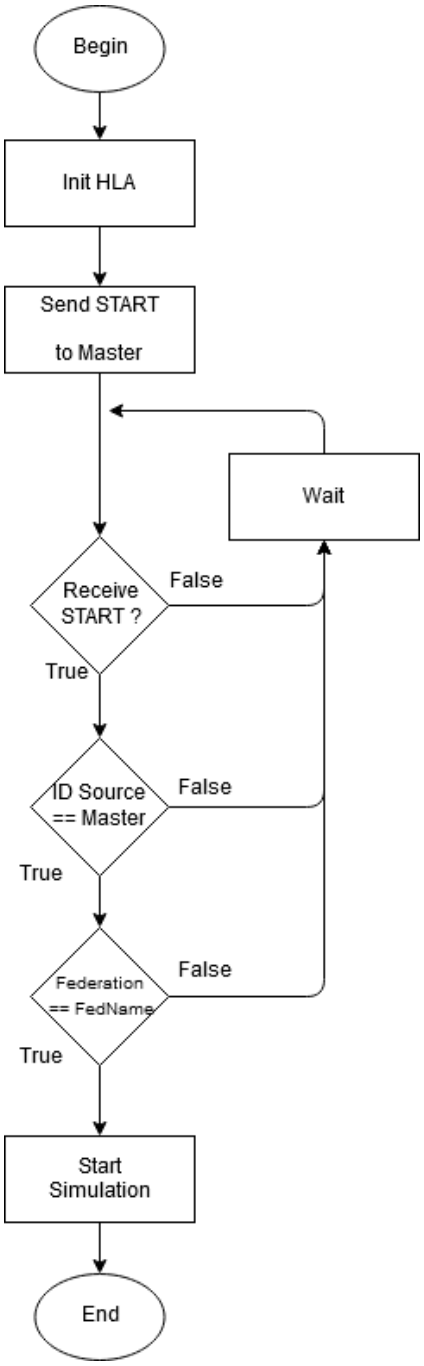


Figure 69 : Algorigramme initialisation fédéré exécutant

De la même façon, le protocole de réception d'un message suit un processus d'authentification basé sur la reconnaissance des identifiants afin de permettre la bonne direction des messages. Comme l'illustre la Figure 70, à la réception d'un message, le fédéré regarde l'identité du destinataire. S'il n'est pas le décideur, le message ne lui est pas adressé. Si par contre, le message provient du décideur,

alors il regarde quel est le destinataire de ce message. Si le fédéré "X" reçoit un message ayant pour identifiant de destination "X", il va exécuter l'ordre contenu dans le message, puis envoyer un accusé de réception (en utilisant le paramètre "Confirm" de l'Interaction Message). Dans le cas où le fédéré "X" reçoit un message ayant pour identifiant de destination "Y", le message sera ignoré.

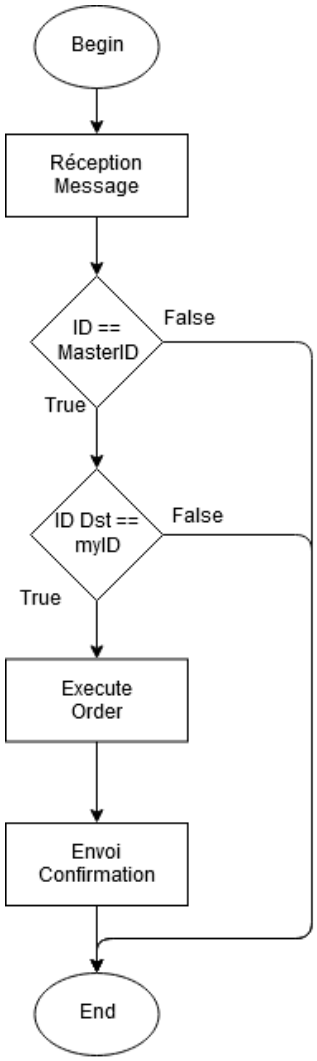


Figure 70 : Algorigramme réception message fédéré exécutant

2.2.Fédéré décideur

Comme pour le fédéré-exécutant, le décideur est avant tout un fédéré normal, sur lequel nous avons encapsulé la couche de gestion BPMN. Cette couche (visible sur la Figure 71) implémente les mêmes mécanismes de communication que décrits précédemment. La différence majeure entre ce fédéré et les autres se situe dans la dernière couche logicielle.

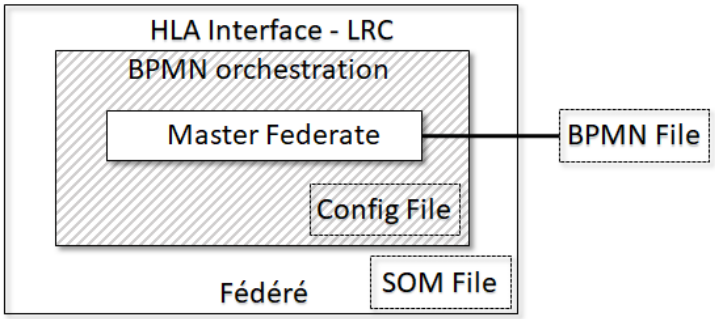


Figure 71 : Vue en couche d'un fédéré « décideur »

Le schéma Figure 71 illustre le contenu d'un fédéré-décideur. Il est pourvu de la même couche BPMN orchestration que ses semblables. La différence se situe au centre de la figure, sur la dernière couche logicielle. Il s'agit d'un fédéré dédié au pilotage des exécutants. On rappelle que le pilotage de fédérés consiste en la capacité de mise en attente, de lancement, et d'attente d'arrêt de ces derniers afin de les diriger comme des boites noires. Le fédéré décideur contient un outil de pilotage capable d'analyser un fichier BPMN, d'envoyer des ordres aux fédérés de la fédération, et d'accéder au fichier de configuration disponible sur la couche inférieure "BPMN Orchestration".

Le fichier BPMN auquel accède le fédéré-décideur a pour rôle de décrire l'ordre d'exécution des fédérés-exécutants. Cet ordre d'exécution va être lu et interprété par le décideur, afin de lancer les fédérés dans un ordre précis. Le standard BPMN étant un héritage de l'UML, son format est le XML. Le fédéré-décideur est donc capable d'analyser l'XML qui décrit le diagramme BPMN de scénario de simulation. Le diagramme BPMN, quant à lui doit respecter un certain nombre d'obligations afin d'assurer le fonctionnement de la simulation distribuée. Pour assurer la liaison entre le BPMN et les fédérés désignés par ce dernier, les noms des tâches doivent correspondre avec les identifiants uniques précisés dans le fichier de configuration (de la couche inférieure).
Ainsi, si la configuration nous indique 4 fédérés avec pour identifiants fed1, fed2, fed3 et fed4, le diagramme BPMN devra décrire le comportement de ces 4 fédérés en employant les identifiants renseignés dans la configuration.

Lors du lancement d'une simulation distribuée, le fédéré-décideur doit être lancé en premier. C'est lui qui va créer la fédération et initialiser la couche BPMN Orchestration. L'algorigramme de la Figure 72 décrit les différentes étapes d'initialisation du fédéré.

Une fois les étapes d'initialisation d'HLA passées, le fédéré lit les informations du fichier de configuration, et vérifie que le fichier BPMN est correctement écrit, et que les noms des fédérés manipulés correspondent aux fédérés du fichier de configuration. Si une erreur apparaît, la simulation n'est pas lancée et l'origine de l'erreur est indiquée dans le rapport de simulation. Suite à ces tests, deux fonctionnements sont possibles. Le mode "hot-plug" (ou "connecté à chaud") lance la simulation sans attendre que tous les fédérés-exécutants soient connectés. A

l'inverse, dans le mode "cold-plug" le décideur va attendre que tous les fédérés soient connectés pour lancer le démarrage de la simulation distribuée.

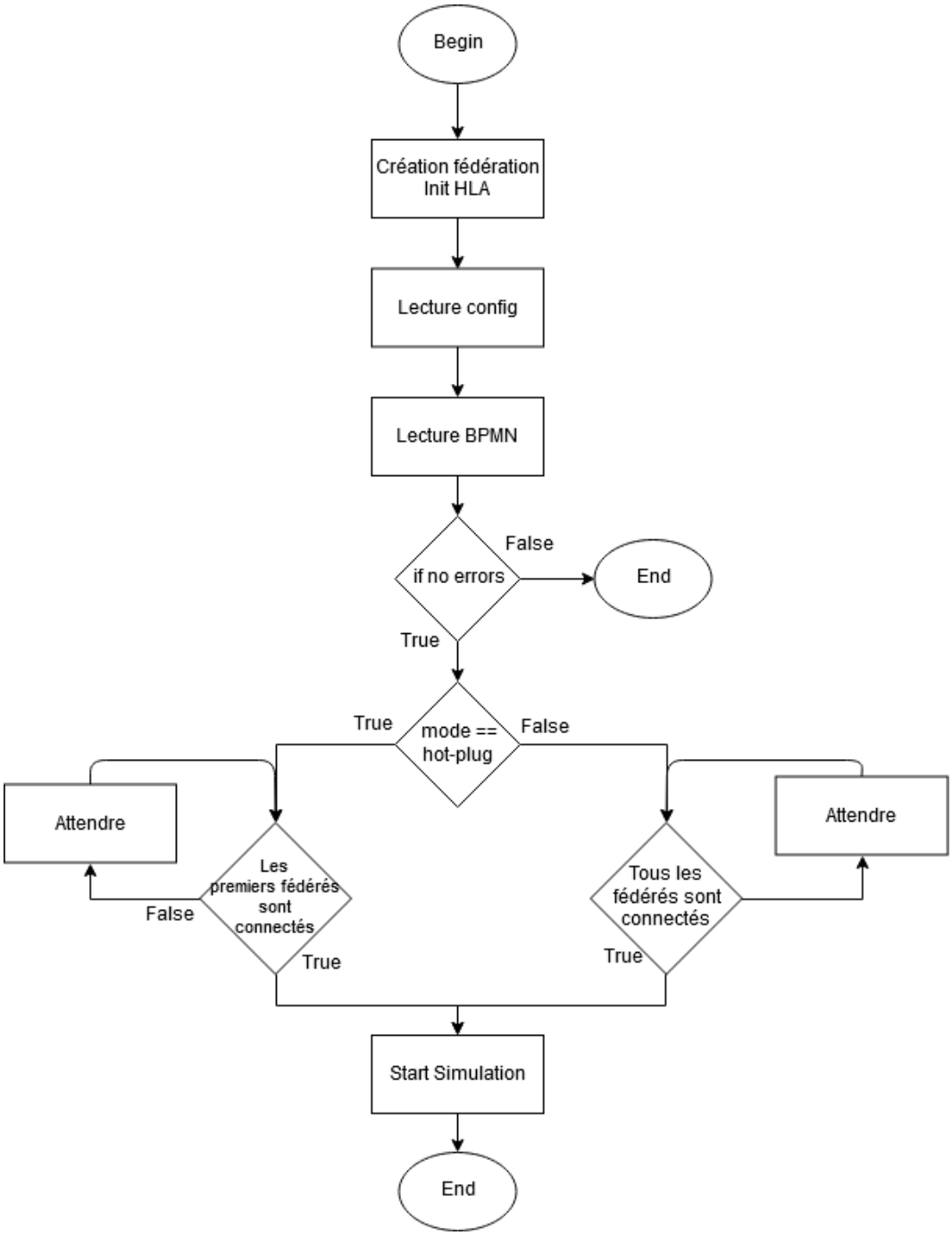


Figure 72 : Algorigramme d'initialisation fédéré décideur

Il existe un dernier mode de contrôle mis au point principalement pour le débuggage de simulation distribuée : le mode console visible sur la Figure 73.

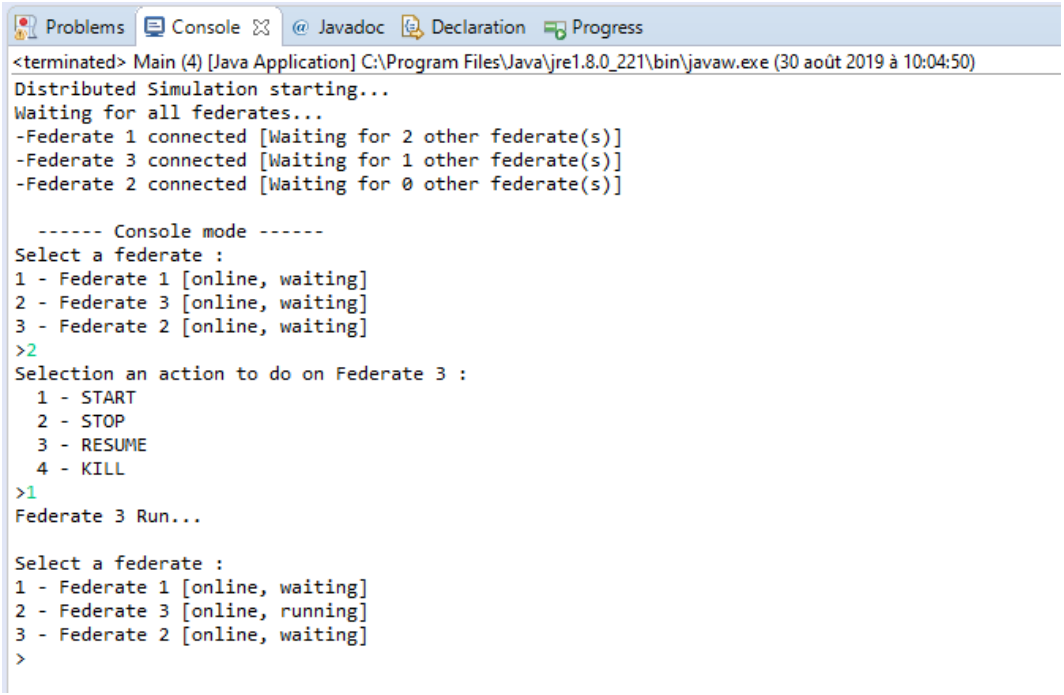


Figure 73 : Contrôle fédération via console

Les étapes d'initialisation du mode console sont les mêmes que le mode "cold-plug". L'ensemble des fédérés décrits dans le diagramme BPMN (ainsi que dans le fichier de configuration) sont attendus avant le démarrage de la fédération. Une fois ces étapes passées, le diagramme BPMN n'est plus utilisé, c'est l'utilisateur qui, via la console, va contrôler l'exécution des fédérés en temps réel. Pour ce faire, comme on peut le voir sur l'exemple de la Figure 73, il va pouvoir choisir un fédéré parmi une liste, et lui envoyer un ordre.

2.3.Mécanisme d'orchestration

Une fois les étapes d'initialisation de HLA et de la couche BPMN orchestration effectuées, la vérification peut commencer. La Figure 74 illustre ce scénario : le décideur parse le fichier BPMN et en fonction du mode de configuration (hot-plug ou cold-plug) il attend les premiers, ou la totalité des fédérés de la simulation distribuée.

Une fois les exécutants connectés à la fédération et sous le contrôle du décideur, le fichier BPMN est analysé à nouveau. Cette fois, les fédérés-exécutants sont lancés dans l'ordre décrit par le diagramme. Une fois un ordre d'exécution lancé, le décideur se met en attente d'un accusé de réception, puis attend l'événement de fin de tâche du fédéré. Il peut alors passer à/aux étape(s) suivantes, et ainsi de suite.

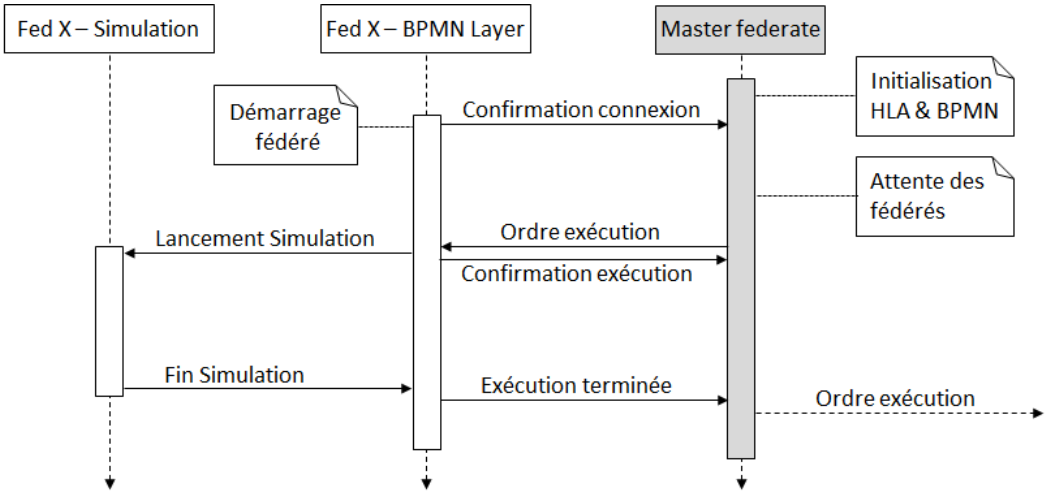


Figure 74 : Diagramme de séquence UML initialisation fédération sous contrôle d'un fédéré décideur

La Figure 75 représente un cas d'utilisation simple afin d'observer le fonctionnement des fédérés selon un diagramme de séquences. On peut observer que le mode de fonctionnement est "cold-plug" car les 4 fédérés ont dû être préalablement connectés pour que le décideur lance la simulation distribuée.

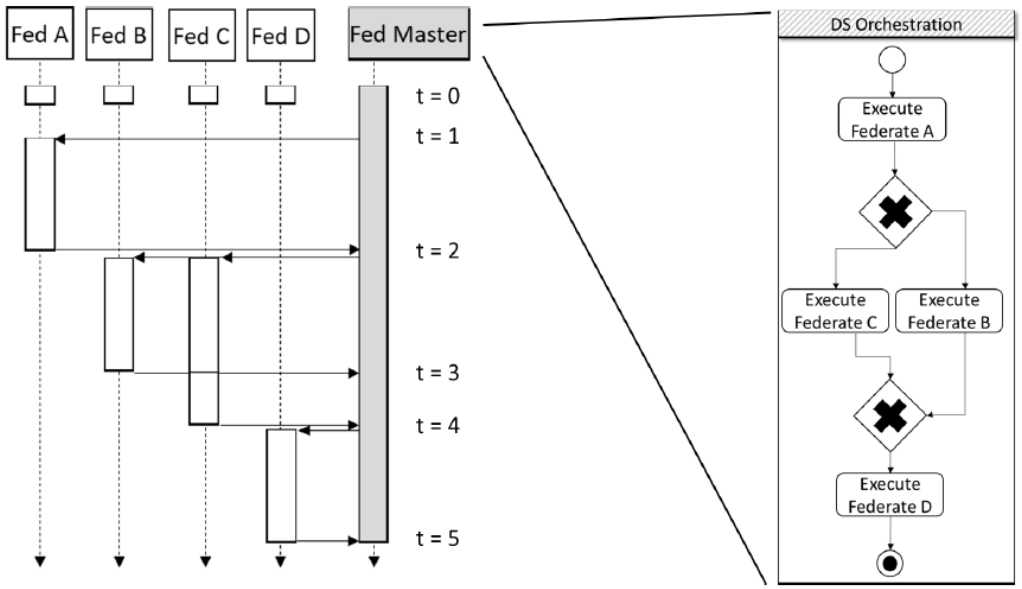


Figure 75 : Diagramme séquence UML + BPMN orchestration

Le fichier de configuration (visible Figure 76) précise sur sa première ligne les différents identifiants des fédérés que l'on peut trouver dans le diagramme BPMN. Ce fichier doit impérativement être le même dans chacun des fédérés (décideur et exécutant). La seconde ligne permet de désigner l'identifiant du fédéré décideur, il sera ainsi reconnaissable par tous les fédérés exécutants. Enfin, la dernière ligne contient le nom de la fédération. Cette information est nécessaire lors de l'initialisation des fédérés.

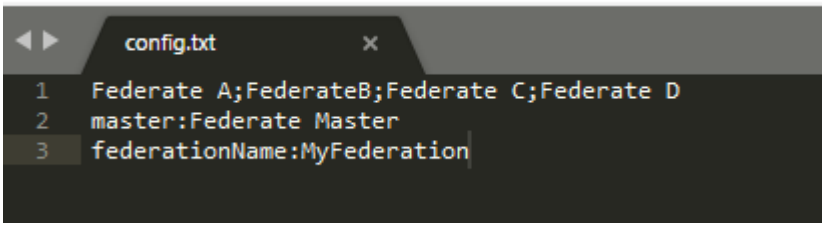


Figure 76 : Fichier de configuration config.txt

3.Applications

3.1.Application générale

Zack Crues, un ingénieur de la National Aeronautics and Space Administration (NASA) a proposé en 2011 à la communauté du Simulation Interoperability Standards Organization (SISO) de lancer un évènement international et interuniversitaire permettant aux participants d'appréhender les différentes problématiques liées à l'interopérabilité des systèmes, tout en se formant à différentes technologies de simulation distribuée.

Baptisé au départ Smackdown, il est devenu Simulation Exploration Experience (SEE). Ce projet est supporté par les milieux industriels, académiques et gouvernementaux (NASA). Il eut lieu pour la première fois à Boston en Avril 2011 lors d'une conférence internationale (SpringSim). Ainsi, des équipes du Massachusetts Institute of Technology (MIT), Florida Institute of Technology (FIT), de l'Université de l'Alabama à Huntsville, de l'université de Pennsylvanie et de l'université d'état de Caroline du Nord (NC State University, Raleigh) ont été rejointes par des étudiants de l'Université de Bordeaux (France) et de l'Université de Gênes (Italie). L'expérience Smackdown est une simulation de ravitaillement Lunaire au cours de laquelle les équipes ont utilisé le standard HLA pour interconnecter des fédérés et reproduire un environnement lunaire.

SISO permis aux élèves d'accéder aux normes HLA tandis que la NASA développa des fédérés de vaisseaux spatiaux et un environnement lunaire. Deux entreprises, MÄK et Pitch Technologies ont fourni aux participants un RTI (Run Time Infrastructure) conforme à la norme HLA. Matlab et Simulink ont également contribué à réduire la courbe d'apprentissage en proposant des échantillons de fédérés préprogrammés. De plus, ForwardSim a développé et fourni un environnement de visualisation 3D permettant un rendu temps réel de la simulation distribuée.

Cet ensemble d'organisations, d'entreprises et d'universités fournissent des outils et des connaissances aux étudiants. Ainsi, ils ont toutes les clefs en main pour construire et alimenter l'environnement de simulation de Smackdown dans le but de planifier, expérimenter, tester, et démontrer leurs compétences techniques, humaines et managériales.

3.1.1.SEE 2018

Durant ce travail de recherche, nous avons eu l'opportunité de participer à trois éditions consécutives du projet SEE. Dans le cadre du SEE 2018, chaque équipe devait développer un fédéré qui constituera la base lunaire. Chaque composant avait pour objectif de communiquer avec le composant d'une autre équipe afin de respecter un scénario de simulation déterminé à l'avance. Nous avons pu mettre en pratique la proposition énoncée dans ce chapitre. L'équipe de l'université de Bordeaux eut l'opportunité de développer un dépôt d'approvisionnement pour des rovers lunaires. Lors de l'édition 2018-2019, nous avons eu l'opportunité d'utiliser la surcouche de simulation distribuée présentée plus tôt afin de piloter deux fédérés tests.

Le contexte de l'événement 2018-2019 consistait en la cohabitation, coordination et la communication de différents fédérés appartement à des équipes différentes. Dans notre cas, nous allons restreindre la description de l'environnement de simulation (Figure 77) uniquement aux fédérés qui sont nécessaires à l'exécution de notre module : le fédéré de l'université de Bordeaux et les fédérés du « Florida Institude of Technology » (FIT).

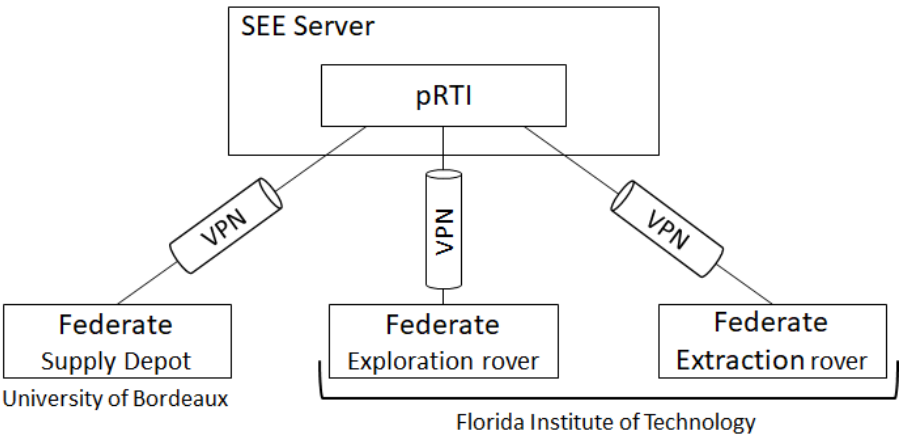


Figure 77 : SEE 2018 Bordeaux and FIT architecture

L'équipe française gérait un dépôt d'approvisionnement (supply depot) doté de différentes fonctionnalités. La première est la communication avec deux types de rover (rover d'exploration et rover d'extraction). La seconde est le stockage d'éléments conteneurisés (apportés par les rovers d'extraction). Enfin il dispose d'une batterie permettant la recharge électrique des deux rovers. Les interactions possibles sont donc au nombre de trois :

- Échange de messages (avec tous les rovers)
- Échange de conteneurs (seulement avec le rover d'extraction)
- Échange d'énergie (avec tous les rovers)

Le dépôt d'approvisionnement n'ayant qu'un seul dock, il ne peut interagir qu'avec un seul rover à la fois. Des algorithmes multi-threadé de gestion de ressources

concurrentes ont donc été mis en place afin de gérer les priorités et une file d'attente entre les rovers.

Ainsi, le fédéré "Supply Depot" assure les fonctions parallèles de :

- Gestion des communications avec les rovers
- Gestion des demandes de docking
- Gestion du docking
- Gestion des interruptions de docking

Chacune de ces fonctionnalités est assurée par un thread du fédéré.

Gestion des communications

La gestion des communications a trois fonctions principales :

- *f.1* - Assurer les échanges automatiques entre rover et station de dépôt
- *f.2* – Autoriser ou non la connexion d'un rover au dépôt
- *f.3* - Assurer une communication textuelle temps réel entre chaque fédéré

Pour cela, après des échanges avec l'université du Florida Institute of Technology, des scénarios ont été conçus et nous avons défini un protocole et une class interaction permettant de gérer tous les cas possibles. Ainsi, l'interaction décrite Figure 78 est placée dans le FOM du fédéré Supply Depot, ainsi que dans ceux des rovers distants :

Figure 78 : Interaction HLA « Message »

L'interaction décrite dans la figure ci-contre permet d'assurer le dialogue entre les acteurs de l'université de Bordeaux et du FIT au sein de la simulation distribuée. Quelle que soit la situation, les protocoles de communication se feront via l'interaction "Message".

Nous pouvons réaliser une liste exhaustive des interactions existant entre chacun des fédérés dans le Tableau 6 suivant.

Cette liste peut être divisée en 3 catégories de fonctions qui sont introduites dans le Tableau 6.

	Fonctions	Paramètre	Supply Depot → Rover	Rover → Supply Depot
f.1	Requête niveau de batterie	Type	"BATTERIE"	"RECIPT"
		Content	""	"49.50"
	Requête identité du fédéré	Type	"ID"	"RECIPT"
		Content	""	"Rover cargo 3"
	Requête contenu du cargo	Type	"CARGO"	"RECIPT"
		Content	""	"49.50"
f.2	Phase d'amarrage	Type	"REQUEST"	"RECIPT"
		Content	- "Wait" - "Dock" - "UnDock"	- "Waiting" - "Docked" - "UnDocked"
		Type		"REQUEST"
		Content		""
f.3	Echange de messages textuels	Type	"MESSAGE"	"MESSAGE"
		Content	"my txt"	"my txt"

Tableau 6 : Interactions possibles entre fédérés

Leur protocole de communication est de type "question - réponse", illustré dans la Figure 79.

La première, *f.1* relève de tous les échanges automatiques entre le supply depot et les rovers externes. Ils ont pour fonction d'assurer la bonne gestion de la flotte des véhicules. Il s'agit de requêtes des niveaux d'énergie restant, d'identité, et d'espace occupé dans les engins lunaires.

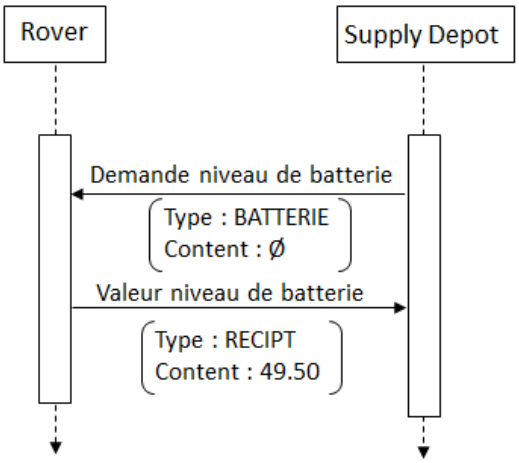


Figure 79 : Exemple de communication entre supply depot et rover

La seconde, *f.2* est utilisée lorsqu'un rover souhaite se connecter physiquement au dépôt d'approvisionnement pour charger ses batteries et/ou décharger le contenu de sa cargaison. Cette demande de docking enclenche le processus de gestion des priorités propres au dépôt d'approvisionnement. Comme nous l'avons décrit plus tôt, cette gestion se fait via 3 processus parallèles, partageant des données, et communiquant entre eux, au sein du même fédéré. On peut prendre comme point d'entrée d'explication de ces processus, la réception d'une demande de docking par le Supply Depot (voir pseudo code Figure 80).

```
DEBUT déclaration variables
Rover monRover // Class représentant le rover en communication ayant pour but
       de se connecter physiquement à la station
Rover  [NB_MAX_ROVER]pileRover  //file  d'attente  des  rovers  avant  liaison
       physique avec la station
FIN déclaration variables

DEBUT fonction réception interaction d'un rover(interaction) //reception d'une
        demande de   docking
       monRover    ←    interaction.RecuperationInformations()    //récupère
              informations d'identité, de niveau de charge et du contenu de
              la     cargaison du rover. Fonctions f.1
       monRover.wait() // on transmet un accusé de réception au rover, on le
              met en attente
       Si pileRover est vide alors
              Ajouter monRover à pileRover
              sinon
              Si monRover.batterie < 5% ET
               monRover.batterie < pileRover[0].batterie ET
               pileRover[0].batterie > 5% alors
                     interruption(monRover)
              sinon
                     ajouter monRover à pileRover
              fin si
              fin si
FIN fonction réception interaction d'un rover
```

Figure 80 : Pseudocode : Réception d'une interaction d'un rover

On constate dans le pseudocode Figure 80 une file d'attente organisée selon une pile qui contient les identifiants des rovers à prendre en charge. Ils seront triés par ordre de priorité. On peut distinguer deux points de sortie dans l'algorithme, qui nous renvoient aux pseudocodes suivants (Figure 81 et Figure 82) :

- L'appel de la fonction *interruption*() qui stoppe la recharge d'un rover en cours pour s'occuper d'un véhicule plus urgent : Figure 82
- L'ajout d'un rover à la pile qui va déclencher l'algorithme de prise en charge des rovers : Figure 81

Le pseudocode de la Figure 81 représente le comportement du fédéré lorsqu'un nouvel élément est ajouté à la liste "pileRover". L'objectif est ici de gérer la recharge batterie et décharger la cargaison des rovers dans la file d'attente. Ce thread attend le remplissage d'une pile partagée (rafraichissement toutes les secondes) qui contiendra des véhicules en attente de prise en charge du dépôt. Lors qu'un véhicule est présent dans la pile, il est pris en charge par l'algorithme. La recharge de sa batterie est faite si son niveau est inférieur à 60%, le déchargement de sa cargaison se fait si le véhicule est un rover d'extraction et que son niveau de charge est inférieur à 50%.

```
DEBUT declaration variables
        Booleen arret ← faux
        Rover monRover
FIN declaration variables

DEBUT fonction attente remplissage pile
        Tant que arret différent de faux faire
                Tant que pileRover est vide faire //attente que le processus
                        précédent place un rover dans la pile
                       Attente(1s)
                Fin tant que


                monRover.sendDock()

                monRover ← lecture information rover(pileRover[0])
                        //récupération des informations batterie et de cargo
                        du rover

                Si monRiver.type == « extraction » &&
                              monRover.cargo > 50% alors
                        decharge cargo(monRover) //décharge cargaison rover
                fin si

                Si monRover.batterie <60% alors
                       recharge batterie(monRover) //recharge batterie rover
                fin si
                liberationRover(monRover) //envoi du signal de libération
        FIN tant que
FIN fonction attente remplissage pile
```

Figure 81 : Pseudocode : gestion d'un rover

Enfin, la fonction d'interruption de la Figure 82 a pour objectif de stopper le dernier processus présenté (fonction attente remplissage pile présentée Figure 81) afin de mettre en charge un rover ayant une priorité supérieure au précédent. Ce thread

peut stopper le précédent, modifier l'ordre de priorité dans la pile, pour ensuite le relancer.

```
DEBUT déclaration variables
        Rover monRover
        Entier itérateur
FIN declaration variables

DEBUT fonction interruption(newRover)

        monRover ← attenteRemplissagePile.getRover()

        Si SupplyDepot est en recharge de batterie alors
                interruption recharge de batterie()
        fin si
        Si Supply Depot est en decharge de cargaison alors
                interruption decharge cargaison()
        fin si

        destruction process attente remplissage pile()

        monRover.sendWait()

                                //decallage pile
        Pour iterateur allant de longeurTableau-1 jusqu'a 0 par pas de -1 faire
                si pileRover[iterateur] n'est pas vide alors
                        pileRover[iterateur+1] ← pileRover[iterateur]
                fin si
        fin pour

        pileRover[0] ← newRover.sendDock()

        attente remplissage pile.start()
FIN fonction interruption
```

Figure 82 : Pseudocode : interruption du processus de gestion d'un rover

Les trois threads que nous venons d'expliciter assurent ainsi les fonctions *f.1* et *f.2* tel qu'on peut le voir sur le diagramme de séquence UML de la Figure 83. Dans la figure, nous avons pris le cas pessimiste du rover arrivant au dépôt les batteries vides (4,5 % de niveau de charge), nécessitant donc une prise en charge immédiate. Ainsi, toutes les fonctions décrites précédemment sont exploitées. La fonction *f.3* permet l'envoi de messages texte entre des différents fédérés.

Quand le rover entre en communication avec le dépôt d'approvisionnement, on suppose que la place est déjà occupée par un autre véhicule. Lors des premiers échanges, le protocole de communication impose une vérification du niveau de batterie : inférieur à 5%, le thread de réception des interactions déclare donc le rover comme étant « prioritaire ». Dans notre exemple, on part du principe que le rover déjà connecté possède un niveau de charge supérieur à 5%, le thread d'interruption déconnecte donc le rover pour y connecter notre nouveau véhicule prioritaire. Le rover précédent, replacé dans la file d'attente sera traité au prochain cycle de prise en charge du thread de gestion du rover.

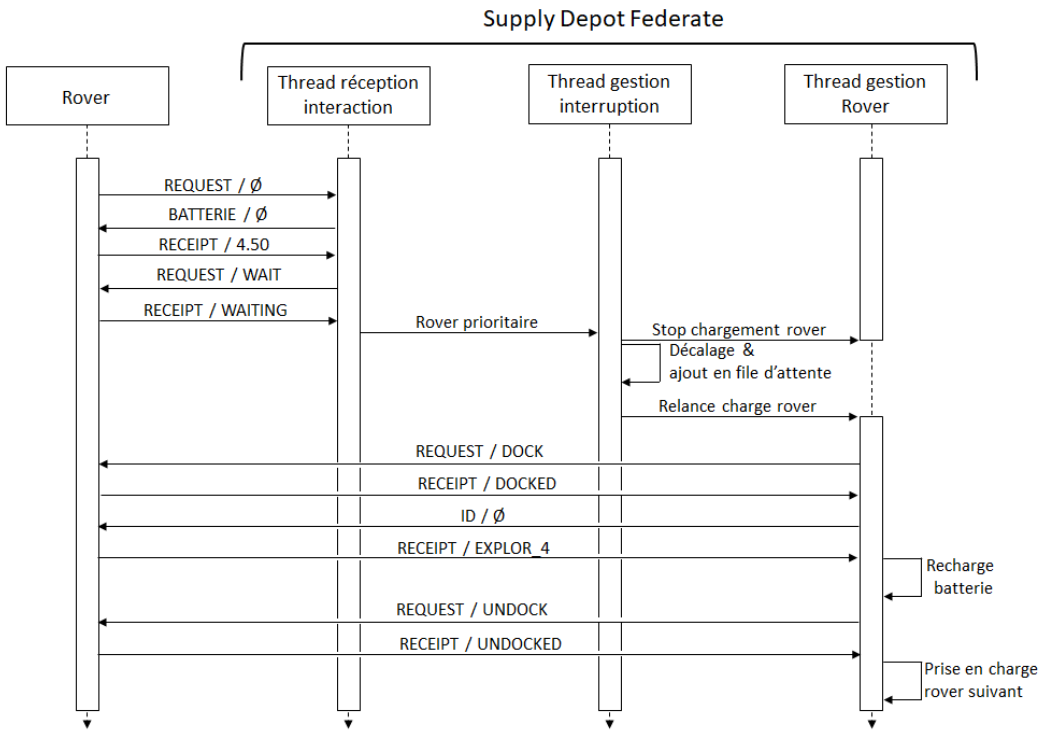


Figure 83 : Exemple de diagramme séquence UML pour un rover ayant un niveau de batterie faible

Enfin, la dernière fonction assurée par le dépôt d'approvisionnement est l'échange de messages textuels en temps réel (*f.3*). En effet, nous avons implémenté un moyen de communication entre les différents rovers et le dépôt d'approvisionnement afin de faciliter la coordination des deux équipes pour assurer le scénario de simulation. Ces échanges ont pour but de permettre une communication sous forme de chat entre l'équipe de l'Université de Bordeaux et celle du Florida Institute of Technology. Les échanges vont se faire via l'interaction décrite par la Figure 78 selon les fonctions décrites dans le Tableau 6.

Afin de garantir les échanges nous avons implémenté un système de messagerie validée par accusé de réception décrit par la Figure 84. Ainsi, un émetteur sait si son message est parvenu au destinataire.

La Figure 84 représente les échanges faits entre deux fédérés dans le cadre d'une communication textuelle temps réel. On peut aussi visualiser sur le graphique la présence de l'utilisateur qui saisit le message. Une fois le message saisi, il est chiffré par une fonction de hachage md5 et mémorisé par le fédéré. Le message est ensuite transmis au destinataire en suivant le protocole décrit par dans le Tableau 6. Le message est donc transmis au destinataire, mais l'expéditeur n'en est pas encore informé. Alors, le fédéré destinataire, chiffre à son tour le message en md5 pour le transmettre à l'expéditeur initial. Une fois reçu, l'expéditeur compare le hash reçu à ceux stockés en mémoire vive afin de valider la communication.

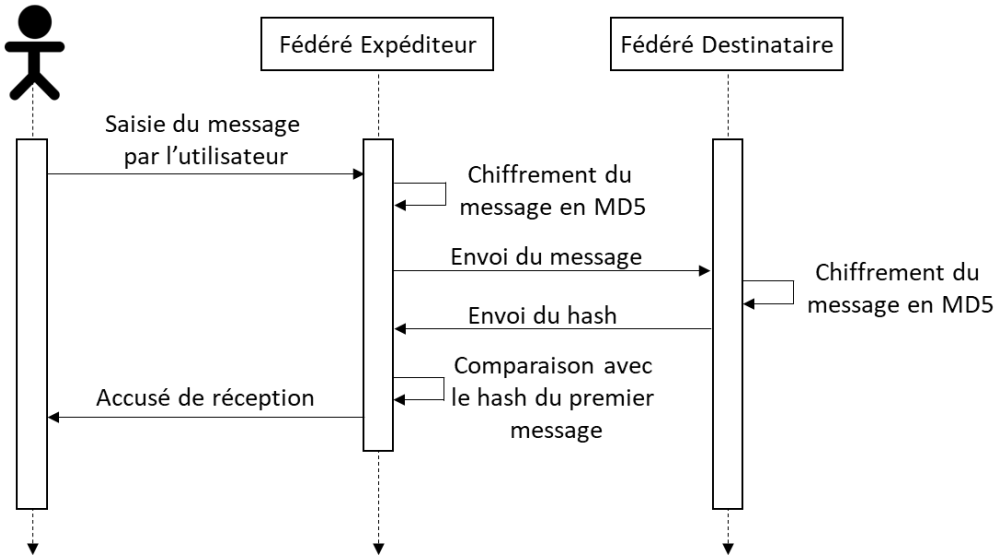


Figure 84 : Echange de messages entre fédérés

3.1.2.Orchestration BPMN dans le cadre du projet SEE

Lors des phases de développement des composants de simulation, chaque équipe travaille seule sur son fédéré. Des phases de tests sont exécutées chaque semaine avec l'ensemble des participants pour mettre au point et vérifier le bon fonctionnement de la simulation dans son ensemble. Entre ces phases, il est nécessaire d'implémenter le composant de simulation en local. Ainsi, le développement des interactions avec les autres fédérés se fait soit uniquement lors des phases de tests hebdomadaires, soit en local en simulant les composants connexes. Dans notre cas, nous avons fait le choix de simuler les rovers du FIT en local.
Ici, la complexité de simulation des rovers ne nous intéresse pas, car nous cherchons uniquement à définir et faire fonctionner un mécanisme de communication à travers les objets et interactions fournis par le standard HLA. Nous avons donc créé deux maquettes simplifiées de rover d'exploration et d'extraction qui seront utilisées uniquement pour implémenter la communication avec le dépôt d'approvisionnement.
L'algorithme des deux rovers est très simple : Quand ils sont exécutés, ils tentent de se connecter au dépôt d'approvisionnement. Afin de faciliter leurs déploiements, on les instancie en passant 4 variables en paramètres :

- Spawn : moment de la simulation ou ils doivent apparaitre
- Type : définit si le rover sera un rover d'exploration ou d'extraction
- Battery : indique le niveau de batterie du rover lors de son apparition
- Cargo (optionnel, uniquement pour les rover de type d'extraction) : indique le niveau de remplissage de la cargaison du rover

Afin de mettre au point et tester l'algorithme du dépôt d'approvisionnement, nous utilisons un dérivé de la surcouche BPMN proposée dans ce chapitre pour piloter et simuler la connexion de rovers en grand nombre au dépôt d'approvisionnement. Comme nous pouvons le voir sur la Figure 85, lors de la simulation locale, seul le

dépôt d'approvisionnement et le fédéré-décideur sont des composants persistants. Un fichier BPMN, descripteur de scénario est donné en entrée du fédéré-décideur, qui va instancier temporairement les rovers dans la fédération comme l'indique le diagramme BPMN.

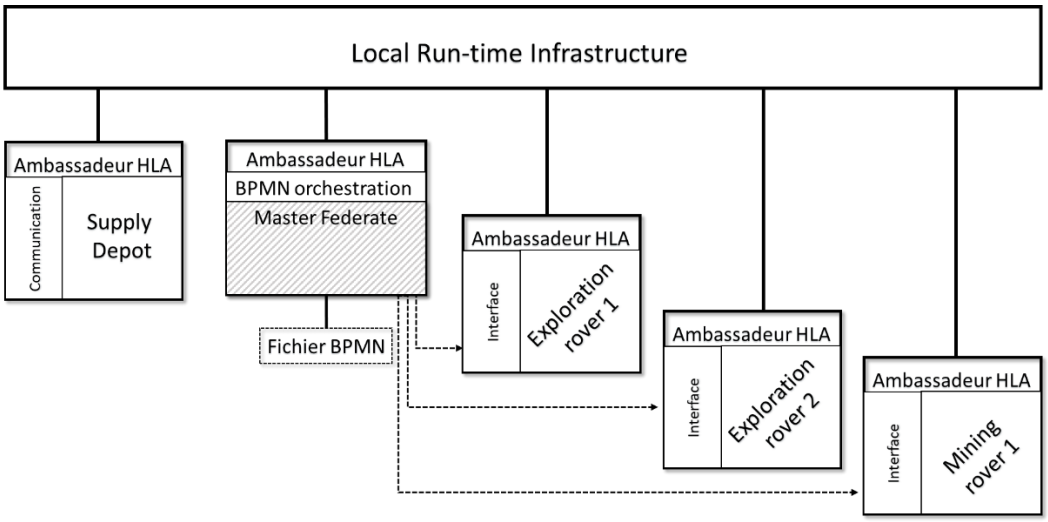


Figure 85 : Architecture test rover local SEE

L'exemple de la Figure 85 illustre trois rovers (deux rovers d'exploration et un rover d'extraction) instanciés dans la fédération par le fédéré-décideur. Pour une simulation avec seulement 5 fédérés, on peut utiliser le fédéré-décideur en mode « cold plug » (voir partie précédente), connecter tous les fédérés, puis lancer la simulation en local. Pour prévoir un scénario plus complexe, nous utiliserons une dérive de mode « hot plug » puisque nous ne lancerons pas les fédérés à chaud, mais c'est directement le décideur qui instanciera les fédérés.

La licence Pitch fournie par le projet SEE nous limite à 10 fédérés connectés simultanément. Autrement dit, le fédéré-décideur ne peut instancier que 8 rovers au même instant (10 fédérés – 1 supply dépôt – 1 fédéré-décideur = 8 fédérés restants).

Pour contourner cette restriction, nous avons modifié le comportement de deux composants :

- Le rover : Après un passage au dépôt d'approvisionnement pour recharger ses batteries et/ou vider le contenu de sa cargaison, quittera la simulation libérant une place pour le prochain composant. Cette action sera précédée d'un message adressé au fédéré-décideur pour l'informer de cette décision.
- Le fédéré-décideur : A l'image du dépôt d'approvisionnement, le fédéré-décideur dispose également une file d'attente. Elle aura pour fonction de ne pas dépasser la limitation des 10 composants simultanés. Avec cette liste d'attente, le composant connait en temps réel le nombre de fédérés présents dans la simulation. A l'instanciation d'un rover, un jeton est ajouté à cette liste. Au retrait d'un rover, un jeton est retiré de cette liste.

La Figure 85 est un exemple de l'architecture que nous venons de décrire. Ici, on constate 3 rovers instanciés par le fédéré-décideur via la lecture du fichier BPMN descripteur de scénario (Figure 86).

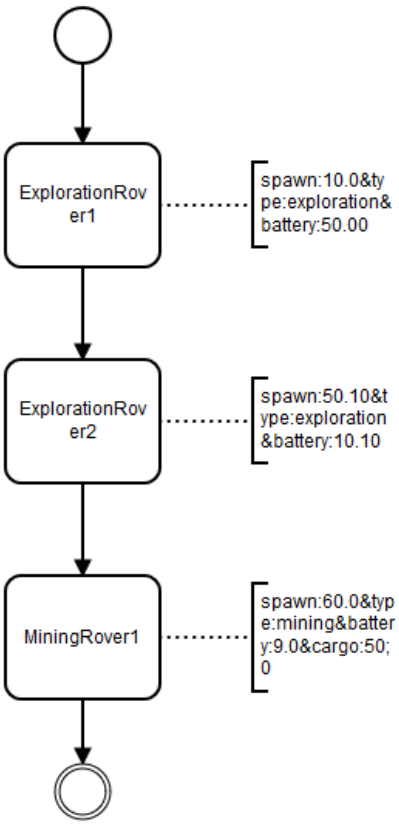


Figure 86 : Modèle BPMN d'apparition des rovers dans la simulation

Le standard BPMN n'a pas pour vocation d'être simulé. En effet, tel qu'il est décrit dans la littérature, il a pour unique fonction la modélisation des systèmes. Dans notre cas, nous utilisons les travaux de [38] qui propose d'implémenter une notion de temporalité dans le standard BPMN en utilisant les commentaires. Dans la Figure 86, les paramètres nécessaires à l'instanciation des rovers seront décrits dans les commentaires de chaque tâche (Spawn, type, mining, batterie, cargo).

Dans l'édition 2019/2020, ces travaux ont été repris par des étudiants de l'IMT Mines d'Alès.

3.2.Application Industrielle (Papyrus)

L'implémentation de l'orchestration d'une simulation distribuée par un modèle Papyrus est une alternative à la solution précédente. Cette proposition permet une meilleure gestion et une meilleure lisibilité des diagrammes, et une facilité d'implémentation accrue.

Afin de définir le contrôle du fédéré-exécutant via Papyrus, nous avons défini une extension du moteur d'exécution Moka inclu dans Papyrus. Cette extension

apporte un profil supplémentaire applicable sur une tâche dans un modèle Papyrus visible sur la Figure 87.

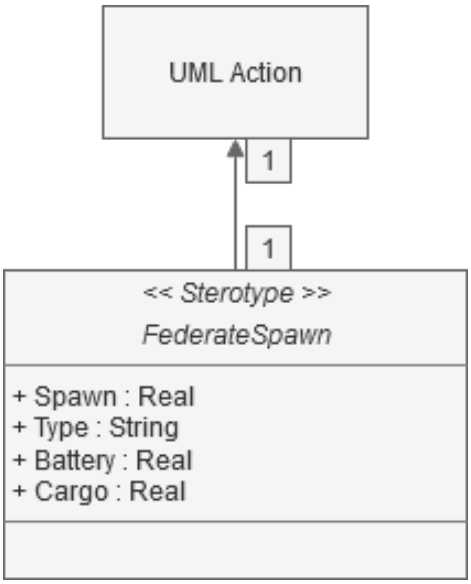


Figure 87 : Profil UML de gestion de fédéré

La Figure 87 illustre le profil UML qui sert de base pour implémenter une extension du moteur Moka afin de gérer un fédéré via Papyrus. Ce diagramme UML nous indique que l'utilisateur peut appliquer un profil UML à une action. Ainsi, via l'interface de Papyrus, il est possible d'ajouter à une tâche, les paramètres « Spawn », « Type », « Battery », et « Cargo » qui seront par la suite interprétés par le moteur d'exécution Moka.

3.2.1.Extension de Moka à la simulation distribuée du projet SEE

L'objectif de cette contribution est d'utiliser l'outil de modélisation intégré à Papyrus pour lancer des simulations distribuées. Pour cela nous avons défini une extension du moteur Moka afin d'implémenter les mécanismes de communication HLA sous l'environnement Papyrus (Figure 88).

Pour ce faire, *EngineExample* appelle la liste des *AdviceFactories* pour ajouter notre propre *factory*. Nous ajoutons donc à la liste, la classe *FederateAdviceFactory*, héritant de *IControllerAdvice*. Cette classe va, lors de la phase d'initialisation des Visiteurs (première étape de la machine à état Figure 27), permettre d'ajouter un *Advice* personnalisé. Cet *Advice* (*FederateAdvice*) est donc ajouté aux *Visiteurs* des tâches faisant référence aux rovers. Pour cette extension de Moka, tous les *FederateAdvice* font référence à un gestionnaire de simulation distribuée (*FederateManager*) qui s'occupe d'instancier et de communiquer avec les fédérés.

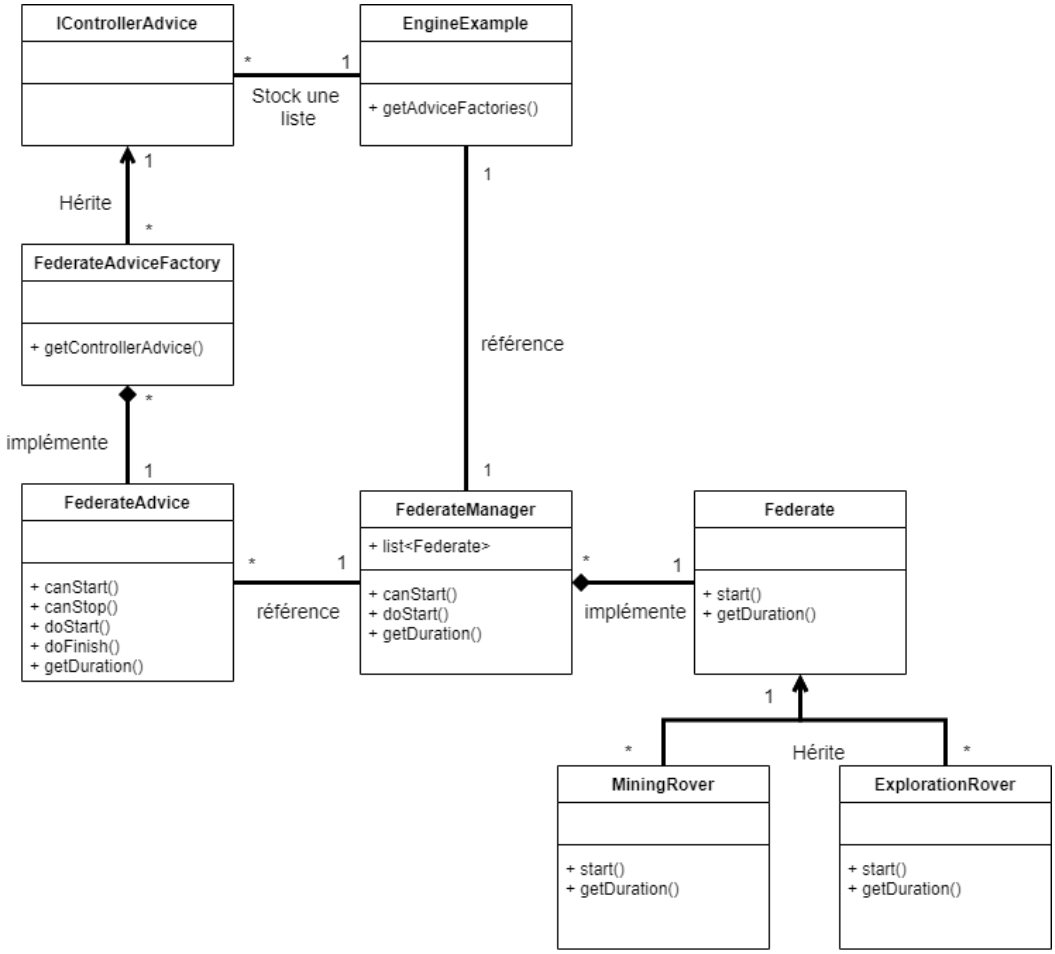


Figure 88 : Diagramme UML Extension Moka pour projet SEE

Lors de l'exécution de la simulation, Moka parcours le graphe UML en dressant la liste des Visiteurs associés à chaque élément UML. Pour chacun de ces Visiteurs, il recense les Advices qui devront être exécutés en temps voulu.

Ainsi, lors de l'exécution, on observe le diagramme de séquence UML suivant (Figure 89).

Dans l'exemple de la Figure 89, nous visualisons l'acteur « Moteur Moka » représentant notre extension du moteur qui hérite du fonctionnement de Moka tout en implémentant un comportement supplémentaire. Cette version du moteur est personnalisée pour le lancement de simulations distribuées. Nous visualisons également un *Advice* associé à une tache UML, cette dernière représentant un fédéré. Enfin, le manager HLA a pour fonction d'initialiser, gérer et terminer une fédération HLA ainsi que ses composants.

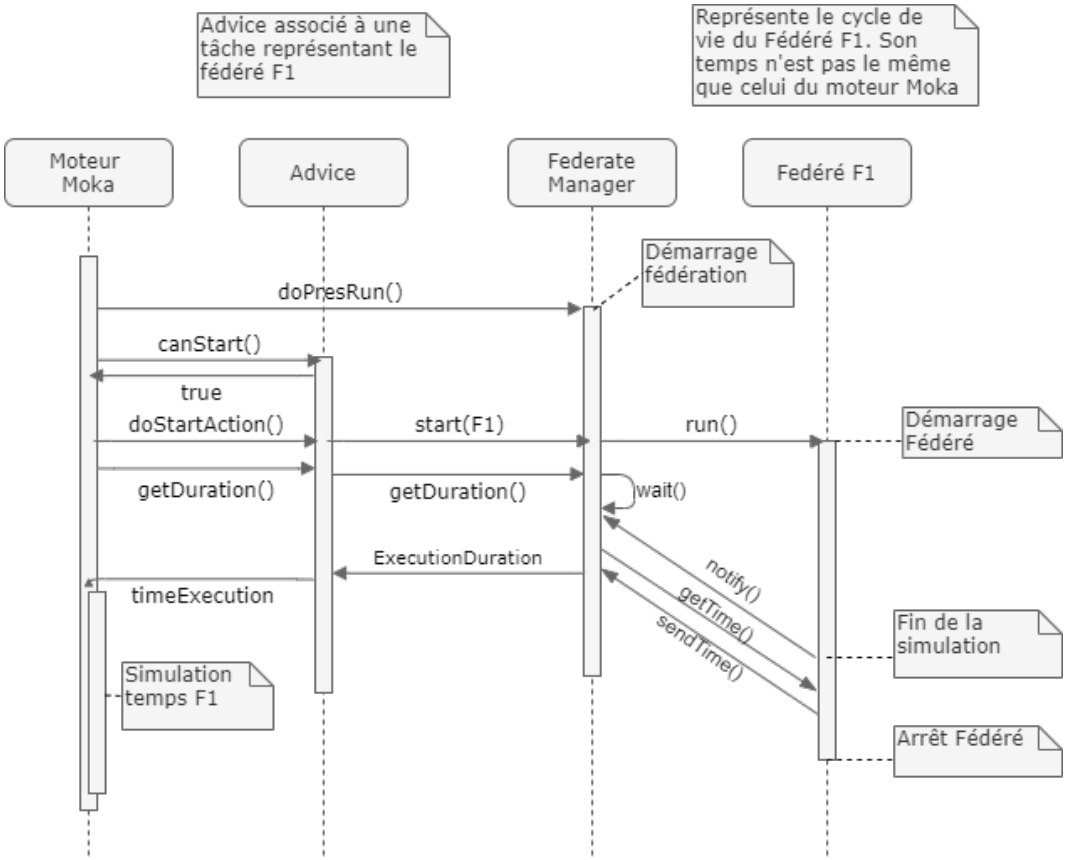


Figure 89 : Diagramme de séquence UML exécution extension Moka

L'exemple donné se place dans le cas d'une première tâche d'un diagramme UML. En effet, on peut lire sur la Figure 89 que la première étape effectuée par le moteur Moka est l'exécution de la fonction *doPresRun().* Nous rappelons que cette fonction est exécutée une fois au démarrage d'une simulation Papyrus. Cette fonction instancie et initialise la classe *FederateManager* qui crée et configure une fédération HLA. Puis, le moteur interroge l'ensemble des Advices associés au(x) premier(s) Visiteur(s). Dans notre cas, il n'y en a qu'un. L'appel de la fonction *canStart()* résulte d'une validation provenant du Visiteur contenant notre *Advice*. La fonction *canStart()* ne peut être bloquante car nous souhaitons lancer les fédérés séquentiellement : rien ne peut bloquer le démarrage d'une tâche, tant que la précédente est terminée. Si tous les *Advices* d'un *Visiteur* valident le démarrage, Moka valide la tâche, la fonction *doStartAction()* de chaque *Advice* est réveillée. Nous utilisons cette tâche pour récupérer les données contenues dans le stéréotype appliqué à la tâche UML. Dans notre exemple, ce stéréotype a été rempli par l'utilisateur lors de la phase de modélisation via l'interface visible sur la Figure 90. Il va contenir le nom du fédéré, son type, son niveau de charge, et son niveau de remplissage s'il s'agit d'un rover d'extraction minière.

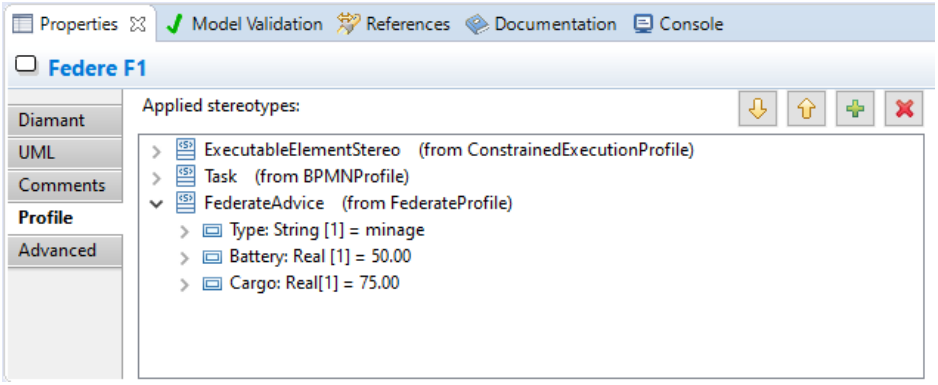


Figure 90 : Interface des profils Papyrus

Une fois les données du fédéré récupérées, on les transmet au gestionnaire HLA qui instancie et lance le composant de simulation distribuée. Après *doStartAction()*, le moteur va demander aux Advices le temps d'activation de la tâche via un appel à la fonction *getDuration()*. Chacun d'entre eux va répondre dans un temps qui sera cumulé. Dans notre cas, on attend que la simulation ait terminé son exécution, puis on interroge l'instance du fédéré afin de récupérer la durée de la simulation (selon son temps propre). Ce temps est converti en temps de simulation Moka puis retourné au moteur. Moka récupère donc le temps que doit durer la tâche et la simule.

Ainsi, l'implémentation d'une extension Papyrus permet de gérer notre flotte de rovers indépendamment et de façon graphique. Dans notre cas d'utilisation, le dépôt d'approvisionnement peut être lancé manuellement à part, ou depuis Papyrus (comme on peut le voir sur la Figure 91). Un des avantages de passer par Papyrus est de pouvoir configurer un scénario de simulation manuellement et de façon graphique.

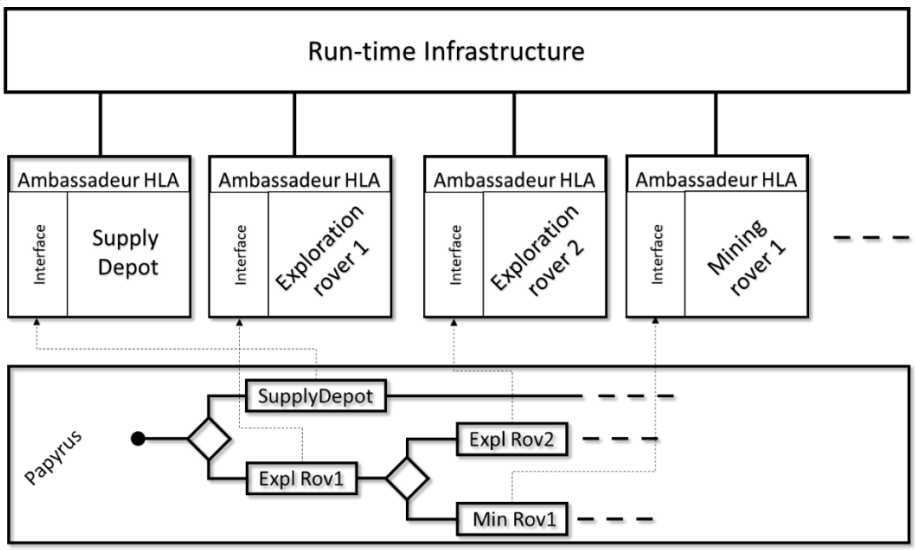


Figure 91 : Modèles Papyrus pour pilotage d'une simulation distribuée dans le cadre du projet SEE

Il est ainsi très simple de configurer, d'exécuter, et d'interpréter les résultats. L'utilisation de Papyrus permet également d'accéder aux résultats de simulation et ainsi, visualiser le scénario de simulation a posteriori (Figure 92).

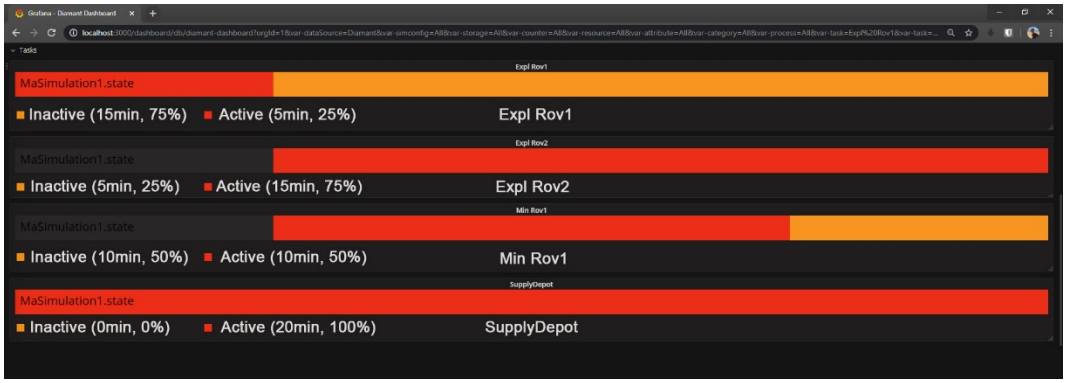


Figure 92 : Résultat de simulation projet SEE via Papyrus

La Figure 92 est un résultat de la simulation exportée dans la base de données influxDB. Les résultats son visualisés avec l'outil graphana qui permet de tracer des graphes, et visualiser des indicateurs de performance.

Ainsi, la gestion d'une simulation distribuée via des mécanismes de décideur/exécutant, et d'outils de modélisation tiers tels que Papyrus permet à l'utilisateur de piloter et visualiser une simulation distribuée de façon simple et graphique.

4.Conclusions

Durant ce chapitre, nous avons proposé une surcouche au standard HLA permettant l'ajout d'un mécanisme de pilotage de fédéré. La première question scientifique que nous nous sommes posée (Q3.1) part du constat que le standard HLA ne permet pas de décrire un scénario de simulation distribuée, ni de le représenter de façon graphique. Pour répondre à cette question, nous avons proposé dans un premier temps une approche via le standard BPMN s'articulant en deux points. Le premier est l'implémentation d'une sur couche au standard afin de définir une hiérarchie entre les différents fédérés. Un fédéré « décideur » va avoir la possibilité d'envoyer des ordres aux fédérés exécutants, leurs permettant d'être orchestrés. Le second point consiste en la définition de diagrammes BPMN illustrant un scénario de simulation distribuée. Ce dernier va être interprété par le fédéré décideur qui, à sa lecture, exécutera les fédérés exécutants suivant l'ordre décrit par le fichier BPMN.
Pour répondre à la seconde interrogation scientifique (3.2), nous avons proposé de développer une extension Papyrus afin de prendre en charge le fédéré « décideur » décrit plus haut. Un outil tel que Papyrus autorisant la personnalisation de son moteur d'exécution, permet de concevoir un scénario d'exécution de simulation distribuée grâce à son outil de modélisation graphique. Cette modification de l'outil permet à l'utilisateur d'éditer des liens entre les composants de la simulation distribuée (les fédérés) et les blocs exécutés par Papyrus.
Dans la suite de ce manuscrit, nous assemblerons les différentes extensions de papyrus proposées au cours des trois derniers chapitres de contribution.

Chapitre 6
Synthèse

Sommaire

1.Introduction

Dans ce chapitre, nous allons détailler le fonctionnement d'une application imbriquant les trois contributions abordées dans les chapitres 3, 4, et 5. L'objectif est de proposer un environnement de modélisation et de simulation hétérogène distribué et dirigé par un processus orchestrateur, tout en ayant une gestion des risques et aléas externalisée. L'outil proposé dans ce chapitre est baptisé par ALSOLENTECH, DIAMANTR.

Le contexte de développement industriel nécessite une bonne compréhension de plusieurs domaines qui apportent des points de vue différents : structurel, fonctionnel, d'interactions avec l'environnement, d'interactions avec les sous-systèmes, etc. L'objectif principal de ce chapitre est de modéliser et de simuler l'ensemble du processus de production de SolR² (cf : Contexte industriel) afin de prévoir et de corriger les erreurs potentielles pouvant se produire au cours du processus réel. Dans cette optique, l'entreprise est représentée par un processus qui interconnecte des blocs de fonctionnalités, ainsi que des interfaces qui assurent une connexion avec l'environnement extérieur. Le système modélisé s'inspire d'une étude de cas fournie par ALSOLENTECH. Le contexte identifié délimite les principaux objectifs à atteindre. Afin de déterminer les besoins de l'entreprise, certains des principaux besoins du système sont identifiés :

- Modéliser et simuler un système de production afin de tester les performances opérationnelles du système simulé sur la base de différents indicateurs (évolution des délais de livraison au cours du temps, évolution de la quantité d'encours au cours du temps, durée du processus simulé).
- Modéliser un système global signifie gérer plusieurs types de ressources (financières, temporelles, matérielles, humaines). L'impact de la plateforme sur l'humain doit être pris en compte, du point de vue de l'entreprise L'outil peut aussi être utilisé pour former les acteurs du projet [140].
- Gérer les risques pendant le processus de simulation afin de modifier le temps de traitement de certaines tâches impactées.
- Se connecter et communiquer avec des outils externes pour une meilleure interopérabilité entre des composants hétérogènes.
- Obtenir des résultats de simulation pour suivre les améliorations des processus simulés.

Dans ce chapitre, nous apporterons une solution à tous les obstacles industriels mentionnés ci-dessus. Chacun de ces obstacles peut être surmonté grâce à un outil ou une plateforme unique fourni par des entreprises et des technologies spécifiques. Les gestionnaires et les décideurs expriment le besoin d'adopter une approche plus pratique et holistique qui permet de traiter tous ces obstacles dans une plate-forme ou un environnement unique.

2.Architecture fonctionelle

Cette contribution fournit un cadre qui intègre toutes les fonctionnalités nécessaires permettant aux entreprises de surmonter les problématiques citées

précédemment. Le système de production général du cas d'étude doit être dans un premier temps modélisé. Chaque activité modélisée peut partager, différentes ressources telles que des moyens humains, des machines, du temps, ou des ressources informatiques (noté H, M, T, IT sur la Figure 93). Le système modélisé peut être régi par plusieurs risques, ou simulations externes. Les systèmes externes ont été développés pour expérimenter les risques environnementaux (contraintes météorologiques, communications avec des outils externes temps réel) ou les risques industriels (incertitudes liées aux connaissances de l'entreprise). La génération de ces risques affecte le système de production de l'entreprise et aide les décideurs à suivre les conséquences de chaque danger. En outre, des événements dangereux seront générés pendant la simulation et les résultats seront visualisés en temps réel sur un outil (log) qui affiche tous les indicateurs clés de performance nécessaires pour suivre l'avancement de la simulation. De plus, un récapitulatif de simulation est également disponible en fin de la simulation. L'outil proposé peut être considéré comme un environnement de formation et d'aide à la décision pour les gestionnaires de projets afin de minimiser les dangers et les pertes de temps, et tester différents scénarios en fonction des modifications apportées au processus.

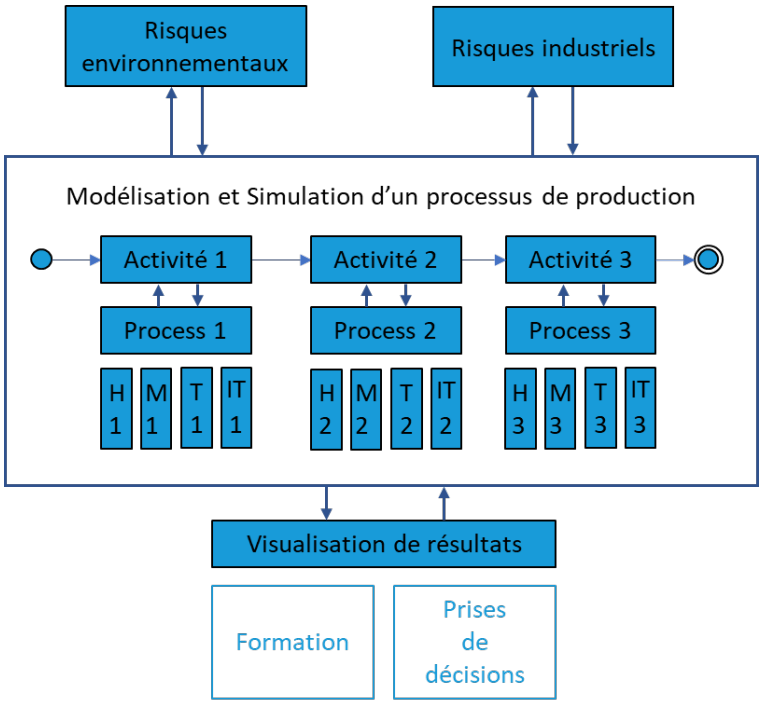


Figure 93 : Architecture conceptuelle

L'architecture conceptuelle proposée est constituée de différents composants issus des contributions exposées précédemment dans ce manuscrit :

- Un outil de modélisation et d'orchestration de composants distribués : Pour cela, Papyrus et son moteur d'exécution Moka seront utilisés pour diriger les différents processus de l'entreprise (voir chapitre 3)
- Des outils de management du risque et des aléas : un composant externe gèrera la dégradation de la simulation au cours de ses exécutions grâce à des outils connexes (voir chapitre 4 et 5)

- Une simulation à évènements discrets : JaamSim sera utilisé pour représenter les processus (voir Figure 93) du cas d'utilisation.
- Un outil d'affichage d'indicateurs de simulation : un composant java assurant l'affichage des informations transitant au sein de la fédération (nous n'avons pas encore détaillé au cours de ce manuscrit)
- Implémentation de simulation distribuée :
    - Un environnement HLA gèrera la communication Papyrus – JaamSim (chapitre 5)
    - Un environnement FMI gèrera la communication Papyrus – FMU et HLA – FMU (chapitre 4)

Comme évoqué précédemment, l'architecture technique de la Figure 94 s'appuie sur les contributions des chapitres 3, 4, et 5. Sa réalisation se base également sur les mêmes principes : la définition d'extensions de Papyrus pour étendre des nouvelles fonctionnalités.

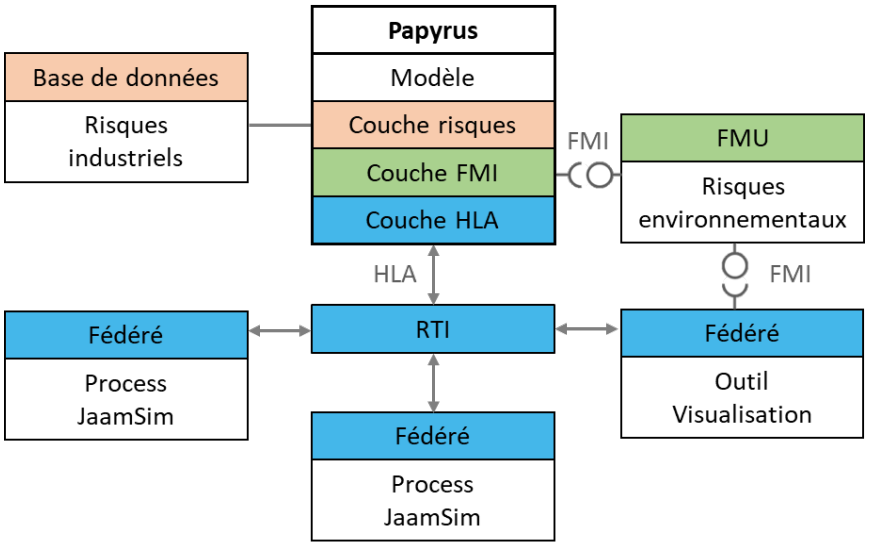


Figure 94 : Architecture technique

Pour constituer l'architecture technique, on observe sur la Figure 94 trois couches logicielles dans l'outil Papyrus :

- La couche orange définit la notion de risque industriel dans un modèle Papyrus et permet les interactions avec une base de données (Excel) contenant les informations sur ces risques industriels (Chapitre 3). Cette notion de risque pourra être accessible aux autres couches Papyrus.
- La couche verte implémente le système de communication FMI avec un composant capable d'interagir avec l'environnement de la simulation dans le but de récupérer des informations sur les risques météorologiques (Chapitre 4).
- La couche bleue permet une communication avec les composants appartenant au standard HLA dans le but de piloter les processus fédérés de l'entreprise (Chapitre 5).

La Figure 94 résume donc l'ensemble des composants qui seront utilisés pour développer la simulation distribuée. Chacun de ces composants sera détaillé dans la suite de ce chapitre et repéré par la couleur de la couche qu'il représente.

3.Structure de l'application de démonstration

3.1.Les composants de simulation

L'outil proposée dans cette section assemble les trois contributions évoquées aux chapitres 3, 4 et 5 de ce manuscrit. Nous allons maintenant détailler chacune des briques qui formeront l'application générale de la Figure 94 baptisée *Diamant++*.

3.1.1.Papyrus

L'ensemble des composants distribués sont destinés à graviter autour de Papyrus qui est l'orchestrateur de simulation. Son rôle est de coordonner les différents outils pour obtenir une plateforme unique et homogène. Son outil de modélisation (extensible par le biais de profils UML), contient le scénario global de simulation faisant référence à des fonctions connexes. Son moteur d'exécution Moka (également extensible), interprète le scénario global, et commande les outils connexes à Papyrus.
Chacune des technologies exploitées par cette contribution (gestion des risques, FMI, et HLA) est traitée par une couche logicielle de Papyrus (profile UML + extension Moka), que l'on empilera pour bâtir la proposition générale illustrée par la Figure 94.

- Couche de gestion des risques industriels

La couche de gestion des risques reprend la contribution décrite dans le chapitre 3. Elle est constituée d'un profile UML ajouté à l'outil de modélisation UML de papyrus, et d'une extension du moteur d'exécution Moka permettant un comportement personnalisé. On rappelle que la fonction d'un profil UML est de personnaliser sémantiquement un langage de base. Ici, nous ajoutons la notion d'identifiant unique (« ID » sur la Figure 95) à chacune des tâches (*Action UML*, Figure 95) du modèle général.

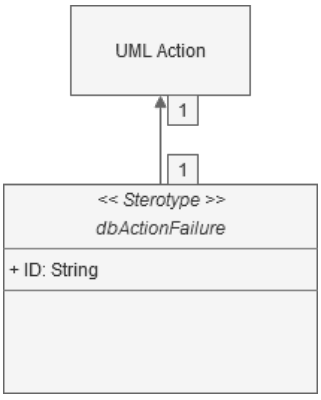


Figure 95 : Profil UML Papyrus : couche gestion des risques industriels

Les informations en matière de risque industriel étant stockées dans plusieurs tables Excel, nous effectuons cette liaison en passant par la définition d'une nouvelle extension Moka. L'EMF Générator d'Eclipse permet d'obtenir les bases de l'implémentation d'une extension Papyrus. L'ensemble des classes et

méthodes définies sur la Figure 96, implémente un *Advice* supplémentaire (au travers de la classe « *AdviceDbActionFailure* ») au Visiteur de chaque tâche étendue par le profil UML « *FailureProfile* ».

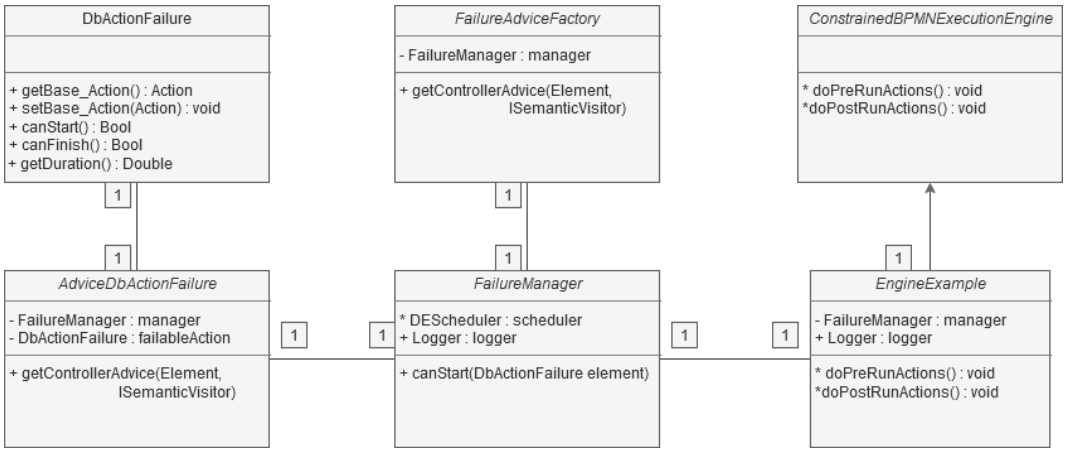


Figure 96 : Diagramme de classes : extension Moka gestion des risques

Lors de l'exécution du modèle, le moteur Moka identifie une tâche en cours d'exécution et effectue une requête horodatée dans la base de données des risques. A la réception de cette requête, l'outil de gestion des risques identifie l'impact des risques courants en fonction du temps simulé et retourne à Papyrus la valeur de l'impact temporel sur la simulation.

- Couche FMI

L'implémentation des communications entre Papyrus et les composants FMU distribués s'établit par le biais d'une couche logicielle Papyrus supplémentaire. La définition de cette extension fait référence au chapitre 4 précédent. Afin de garantir une communication entre l'outil Papyrus et l'environnement de simulation mis à disposition par le standard FMI, nous ajoutons trois composants au système :

- **Un profil UML** (qui viendra se superposer au précédent) permettant à un utilisateur de définir une liaison entre une tâche du diagramme Papyrus et un FMU. Le paramètre « *Path* » de la Figure 97 contient le chemin d'accès au fichier FMU à exécuter.
- **Un fichier de configuration Papyrus** contenant les informations complémentaires sur la nature des FMU pris en charge par papyrus. Un fichier de configuration json est accessible à la racine de Papyrus, et décrit les données relatives aux composants externes manipulés par l'outil. Dans notre cas il contiendra la nature et la valeur des paramètres d'entrée des fichiers FMU chargés par l'outil, ainsi que les valeurs de sortie des composants.
- **Une extension Moka** (décrit par la Figure 97) récupérant d'une part le chemin d'accès du FMU disponible dans le profil UML, et d'autre part les différentes informations relatives au composant FMU dans le fichier décrit précédemment.

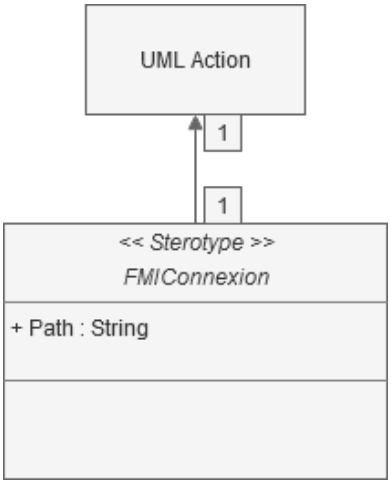


Figure 97 : Profil UML Papyrus : couche gestion des composants FMI

L'ensemble de ses ajouts est fait via la déclaration d'un *Advice* supplémentaire à l'architecture de Papyrus (« *AdviceDbActionFMULoader* » sur la Figure 98) intégrant les bibliothèques java capables d'interagir avec un composant FMU : JavaFMI.

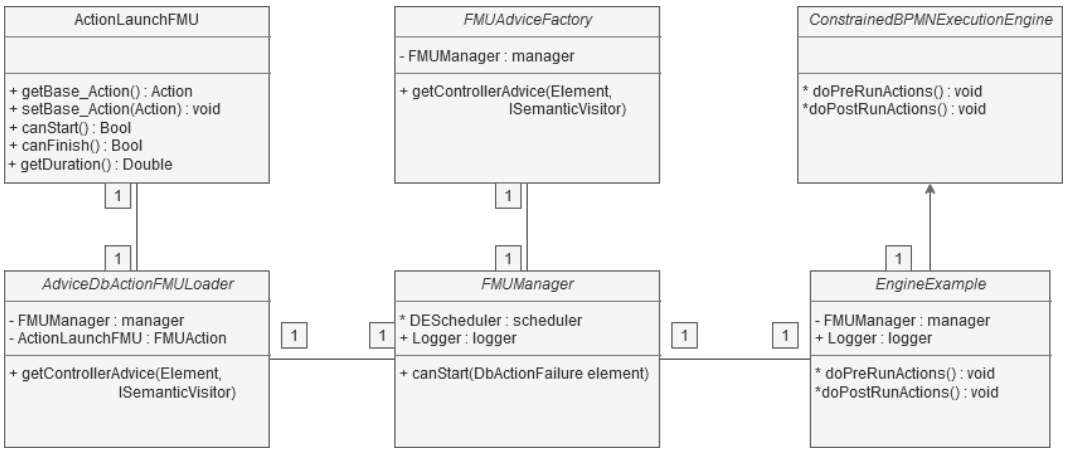


Figure 98 : Diagramme de classes : extension Moka pour FMI

Lors de l'exécution d'une tâche étendue par le profil décrit, le moteur Moka effectue une requête d'avancement d'un temps (valeur de temps définie par le fichier de configuration) auprès du FMU indiqué par le champ « Path » de la Figure 97. A la réception de cette requête, le FMU avance son temps propre, et retourne des valeurs à l'extension, qui sont inscrites dans le fichier de configuration du FMU. Depuis un modèle Papyrus, on peut donc commander l'exécution de composants compatibles FMI.

- Couche HLA

La couche Papyrus permettant le pilotage d'une fédération HLA fait référence à la contribution du chapitre 5 précédent. Elle reprend la proposition d'une surcouche appliquée aux fédérés afin d'implémenter un mécanisme décideur / exécutant entre chacun des éléments du réseau. Pour ce faire, comme décrit dans l'application industrielle, il est question d'utiliser non pas le langage BPMN depuis

un fédéré indépendant, mais d'utiliser l'outil de modélisation intégré à Papyrus pour orchestrer des fédérés connexes.

Comme pour les extensions précédentes, cette couche Papyrus passe par la définition d'un profil UML personnalisé et appliqué aux tâches Papyrus représentant un composant HLA. Ici, nous déclarons un nouveau stéréotype qui sera appliqué aux actions UML et qui contiendra un champ *FederateName* (voir Figure 99) afin d'identifier le fédéré qui sera lancé par Moka au moment de l'exécution.

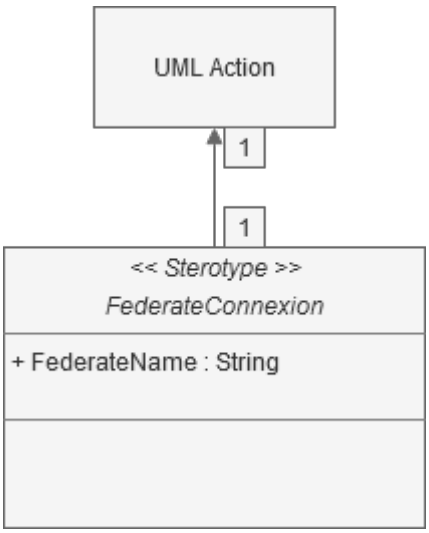


Figure 99 : Profil UML Papyrus : couche gestion des composants HLA

Au lancement d'une simulation, la fonction *doPreRunActions()* de l'*EngineExample* (voir Figure 100) est exécutée. Moka instancie la classe *MasterFederate* qui permet de créer une fédération préconfigurée par le développer (définition du nom de la fédération, passage des fichiers FOM, nombre de fédérés attendu). Moka se met en attente de la connexion de tous les fédérés nécessaires à la simulation pour pouvoir aller plus loin. Une fois le bon nombre de fédérés connectés à la simulation, le modèle Papyrus est exécuté jusqu'à atteindre une tâche étendue par le profil UML signifiant le lancement d'un fédéré (Figure 99). Alors, le *Visiteur* de la tâche UML interroge les fonctions *canStart()* de chaque *Advice* qui lui sont associés. Dans notre cas, *AdviceDBActionFederateLoader* va se référer au *FederateManager* qui lancera le fédéré associé au champ *FederateName* décrit par le profil UML en passant par la classe *MasterFederate*. En d'autres termes, le fédéré décideur donne l'ordre d'exécution à un fédéré exécutant, qui lui, est passé en paramètre.

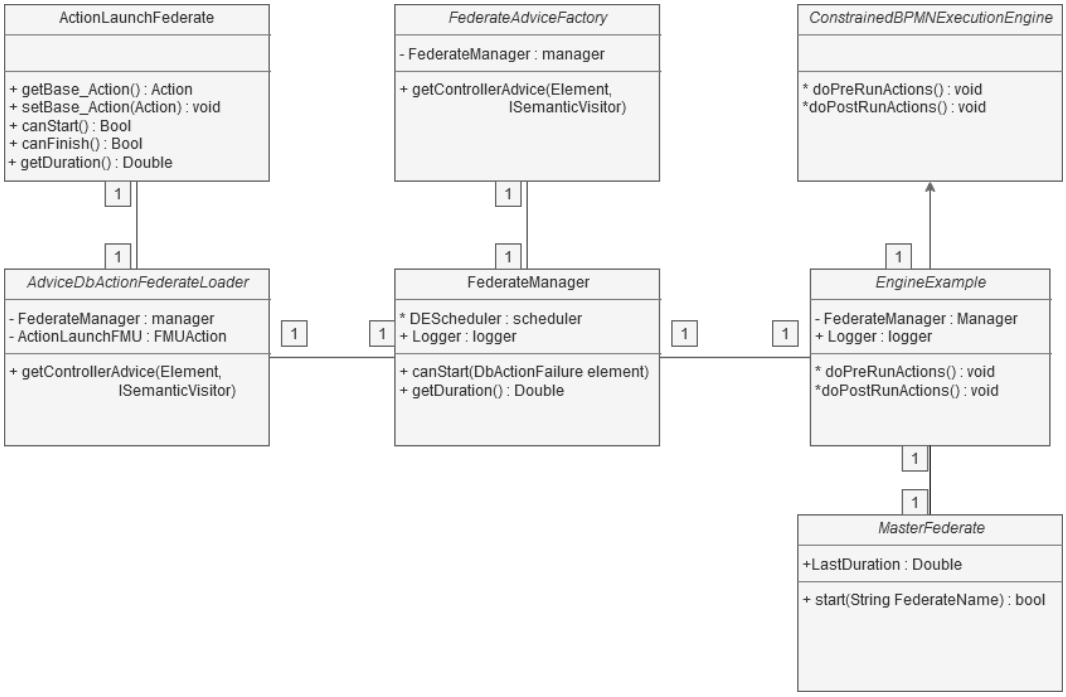


Figure 100 : Diagramme de classes : extension Moka pour HLA

Dans un second temps, Moka interroge la fonction *getDuration()* de chaque *Advice*. On rappelle que cette fonction doit retourner le temps cumulé simulé de tous les Advices de la tâche en cours. A l'exécution de cette fonction dans l'*AdviceDbActionFederateLoader*, le *federateManager* met en attente le moteur Moka et boucle sur le changement d'état de la variable *LastDuration* de la classe *MasterFederate*. De son côté, la classe *masterFederate* est en attente (souscrit) de l'interaction « *FederateEnd* » décrit par la Figure 101.

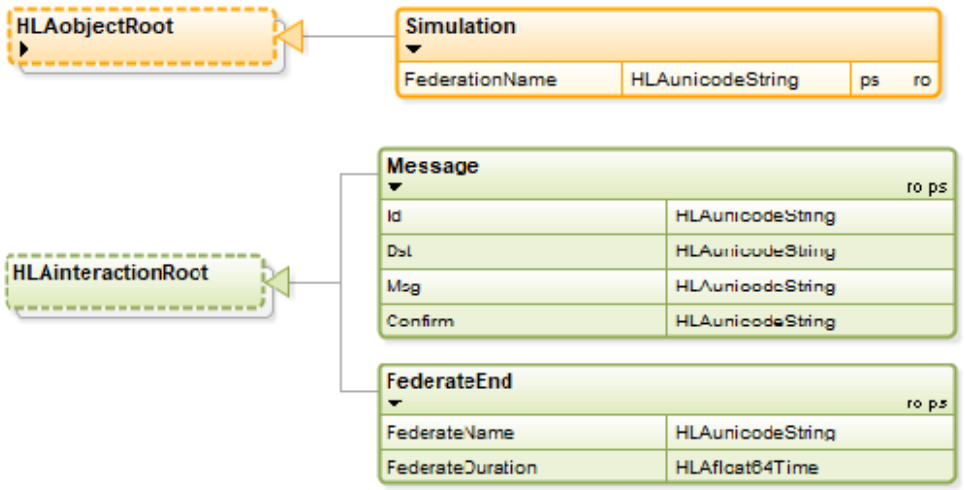


Figure 101 : HLA Objets et Interactions pour l'orchestration Papyrus

L'interaction *FederateEnd* est envoyée par le composant distribué une fois qu'il a terminé sa tâche. Il l'envoie en message à destination de Papyrus pour lui indiquer que son opération est terminée, et lui transmettre son temps de simulation. Ce temps sera récupéré par le *MasterFederate* et enregistré dans une variable de classe *LastDuration*. *FederateManager* sera alors en mesure de le récupérer pour

le transmettre à l'ordonnanceur Moka pour représenter la durée de la tâche distribuée.

Le fichier de configuration de simulation contiendra un ensemble de variables à transmettre au fédéré pour son bon fonctionnement.

3.1.2.Outils de gestion des risques (industriels et environnementaux)

Dans le contexte de la simulation du processus industriel, la gestion des risques est mise en place au travers de deux outils différents :

- La gestion des risques industriels se fait via la communication entre Papyrus et un fichier Excel présenté dans le chapitre 3 précédent.
- La gestion des risques environnementaux se fait via la communication entre Papyrus et un FMU présenté dans le chapitre 4 précédent.

La gestion des risques est gérée par deux modules conçus pour inclure un traitement des risques dans la simulation du système de production. Cette externalisation permet aux ingénieurs de modéliser le système sur un outil, et de définir les règles qui auront un impact sur le système dans des modules séparés. La gravité d'un risque est observée ou calculée par des experts, puis décrite par des équations en fonctions du temps. L'objectif est de mettre en place un système adhoc connecté communiquant avec le moteur de simulation (Moka) pendant l'exécution de la simulation. Ce système peut insérer des pannes ou ralentissements dans le modèle principal de Papyrus impactant la durée d'une simulation.

Comme nous l'avons évoqué plus tôt, cette application regroupe deux outils de gestion des risques. Le premier est un ensemble de tables Excel regroupant des équations définies par le contexte industriel. Elles fonctionnent de façon synchronisée avec Papyrus par un système d'identifiant. L'extension de cette gestion des risques (IDTask de la Figure 95) permet d'ajouter un identifiant unique à une tâche Papyrus. Lors de l'exécution de la tâche concernée, Papyrus inscrit dans le tableau Excel la date courante de la simulation, et met à jour l'ensemble des équations des tables Excel. Puis, il récupère la valeur retournée par le tableau pour l'identifiant donné (module orange sur la Figure 102).

Le deuxième outil concerne la gestion des risques environnementaux. Cet outil fait référence à la contribution du chapitre 4 puisqu'il s'agit d'un composant FMU relié à Papyrus via la couche de communication FMI. Le FMU est capable d'effectuer séquentiellement des requêtes d'informations météorologiques en fonction de paramètres placés dans le fichier de configuration de la simulation (module vert sur la Figure 102).

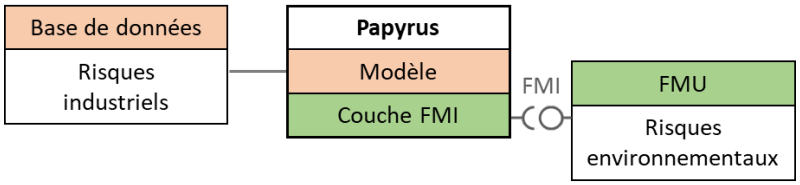


Figure 102 : Outils de gestion des risques industriels et environnementaux

La couche HLA décrite précédemment n'est pas employée pour la gestion des risques. Son rôle va être de coordonner des blocs comportementaux du système de production distribué par un outil de modélisation et simulation : JaamSim.

3.1.3.JaamSim

On rappelle que JaamSim est un logiciel de simulation libre, basé sur Java et développé par une société d'ingénierie australienne nommée Ausenco. La principale caractéristique qui différencie JaamSim des autres outils de M&S à événements discrets du marché est que les utilisateurs peuvent concevoir et fabriquer leurs propres palettes d'objets. Lorsque JaamSim est lancé, une interface graphique apparaît.
Les utilisateurs peuvent utiliser cette interface graphique pour ajouter des entités et créer le modèle de simulation, ou écrire/modifier un fichier de configuration (.cfg) dans lequel toutes les entités/objets peuvent être ajoutés et configurés. Certains utilisateurs peuvent préférer l'interface graphique pour glisser et déposer leurs entités et les configurer sur l'interface utilisateur graphique (GUI). D'autres, en particulier les programmeurs, trouveront plus rapide et plus facile de créer ou modifier le fichier de configuration (.cfg). Ce logiciel est utilisé dans nos travaux de recherche, en raison de sa transparence, de sa fiabilité, et de sa récente adaptation au standard HLA [141].

JaamSim n'a pas été conçu à l'origine pour les communications vers des systèmes externes et n'est pas adapté à la DS ; il est considéré comme un simulateur boîte noire. Cette contribution utilise une version étendue de JaamSim [141] qui l'ouvre à la simulation distribuée via le standard HLA. Dans [141], l'auteur propose d'éditer le code source de l'outil dans le but de le faire communiquer avec différentes instances de JaamSim via le standard HLA. Nous utiliserons cette proposition dans le but de faire collaborer JaamSim avec un systèmes externe tel que Papyrus. Nous avons modifié cette version pour l'adapter à notre contexte d'étude.

3.1.4.Fonctionnement JaamSim

La Figure 103 représente un modèle de supply chain conçu avec l'outil JaamSim. L'interface graphique permet la création de processus industriels avec l'aide d'une palette d'outils.

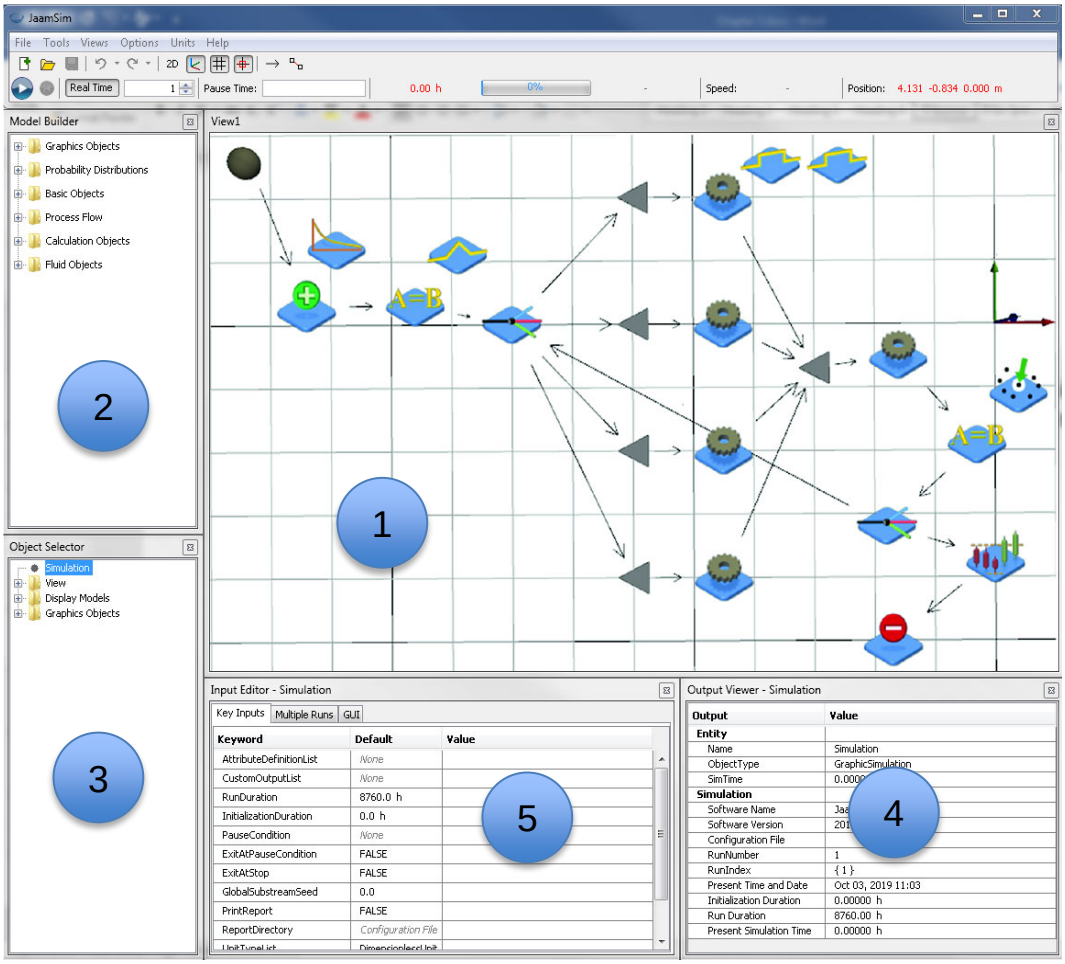


Figure 103 : Interface graphique JaamSim

On peut diviser l'interface graphique de JaamSim en 5 compartiments :

- (1) View : Contient la modélisation de l'utilisateur. L'utilisateur dépose les composants disponibles dans le Model Builder afin de représenter l'agencement de ses composants 2D ou 3D.
- (2) Model Builder : L'utilisateur peut y trouver différents composants et choisir les entités nécessaires à la construction du modèle de simulation. Dans la palette du générateur de modèle, l'utilisateur peut choisir différents objets graphiques, des distributions de probabilité, des objets de base, des objets liés au déroulement du processus, des objets de calcul, et des objets en rapport avec la mécanique des fluides.
- (3) Object Selector : Arborescence des différents composants utilisés dans le projet courant
- (4) Input Editor : Chaque composant dispose d'un ensemble de paramètres éditables par l'utilisateur pour complexifier son modèle
- (5) Output Viewer : Cette fenêtre permet à l'utilisateur de visualiser l'état des variables propres à chaque composant de son modèle pendant la modélisation ou pendant la simulation

C'est à partir de cet ensemble d'outils que deux processus industriels de notre contexte ont été modélisés (nous décrirons ces deux processus plus tard dans ce chapitre). Afin de comprendre les modèles, et les mécanismes de communication

HLA implémentés dans l'outil, il est nécessaire de détailler le fonctionnement de certains composants :

- SimEntity : L'objet SimEntity est l'entité de base des modèles de flux de processus. La principale caractéristique d'une SimEntity est qu'elle peut être attribuée à différentes étapes du flux de processus. Les objets du flux de processus peuvent l'accueillir, la faire attendre ou la transférer à un autre objet. On peut définir un ou plusieurs attributs à une SimEntity. Un attribut fonctionne comme un attribut public de classe Java : il est déclaré comme une variable et est accessible par les autres membres du système. Ces ensembles d'attributs sont définis par une « AttributeDefinitionList » dans l'input Editor d'une SimEntity comme présenté dans la Figure 104.

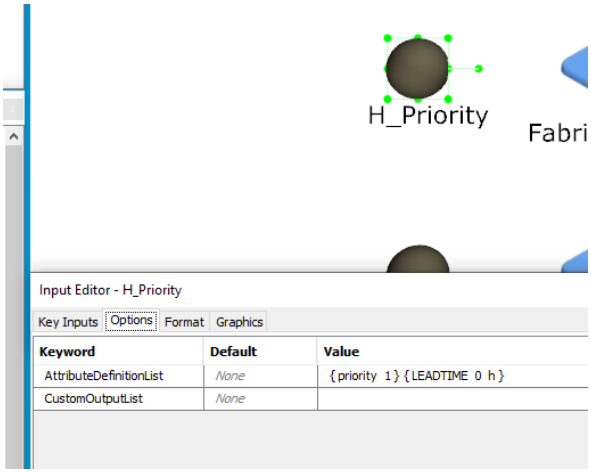


Figure 104 : Exemple création attribut d'entité

  Ici, on affecte à l'entité « H_Priority » deux attributs : « priority » de type entier défini à 1, et « LEADTIME », type temporel défini à 0 heures.
- EntityGenerator : Composant ayant la capacité de générer des instances d'une SimEntity. Souvent en début de flux, il crée l'objet, et l'introduit dans un flux de processus.
- Server : L'objet Server traite une SimEntity. Il la reçoit d'un composant, a la capacité de la mettre dans une file d'attente avant de la passer au composant suivant.
- Assign : Composant permettant d'effectuer un ou plusieurs calculs sur les attributs des objets de la simulation à chaque fois qu'il reçoit une entité entrante. C'est le seul endroit où la valeur d'un attribut peut changer.

L'ensemble de ces composants permet la modélisation de flux industriels. Il est à noter que cette liste n'est pas exhaustive : elle résume les principales fonctions nécessaires à la compréhension des modèles et à l'articulation avec Papyrus par le biais d'HLA.

3.1.5.JaamSim et HLA

Afin de rendre JaamSim compatible avec le standard HLA, certaines modifications ont été apportées au code de ce dernier. Pour créer un fédéré contenant une instance de JaamSim, il est nécessaire d'encapsuler l'outil dans une classe « Federate » effectuant l'ensemble des initialisations HLA. Une fois le fédéré

connecté à la fédération, il va se mettre en attente de la réception de deux informations (voir Figure 105) :

- Recevoir une mise à jour des attributs de l'objet « scenario » qui indiquera les paramètres d'entrée à la simulation JaamSim
- Recevoir une interaction du fédéré décideur (Papyrus) lui ordonnant le démarrage de la simulation (voir chapitre 5).

Figure 105 : Objets nécessaires à la communication JaamSim - Papyrus

Suite à la réception de ces deux informations, le fédéré démarre la simulation JaamSim.

Dans le code source de JaamSim, chacun des composants de la palette d'outils possède sa propre classe Java. C'est via ces classes que nous récupérons des informations de la simulation pour les rendre accessibles sur le canal de communication HLA. Dans notre cas, les paramètres de sortie de la simulation sont trois indicateurs de performance, définis comme des paramètres dans la simulation JaamSim :

- L'évolution des délais de livraison (représentée par le paramètre LEADTIME)
- L'évolution de la quantité d'encours (représentée par le paramètre Work In Progress : WIP)
- La durée totale d'une simulation (représentée par le paramètre SIMTIME).

Ces trois indicateurs sont voués à être envoyés au travers de la fédération, et sont donc déclarés dès le début dans le fichier FOM. Durant la simulation JaamSim, ils sont calculés à partir du composant « Assign ». C'est donc via la classe Assign que nous récupérons ces indicateurs pour les publier à la fédération. La Figure 106 est un diagramme UML représentant les classes nécessaires au fonctionnement de HLA au sein d'une instance JaamSim. La classe « Federate » initialise la liaison avec le RTI et la fédération. Une fois l'ordre d'exécution reçu (par Papyrus), la classe « GUIFrame » est instanciée ce qui a pour effet de lancer la simulation JaamSim.

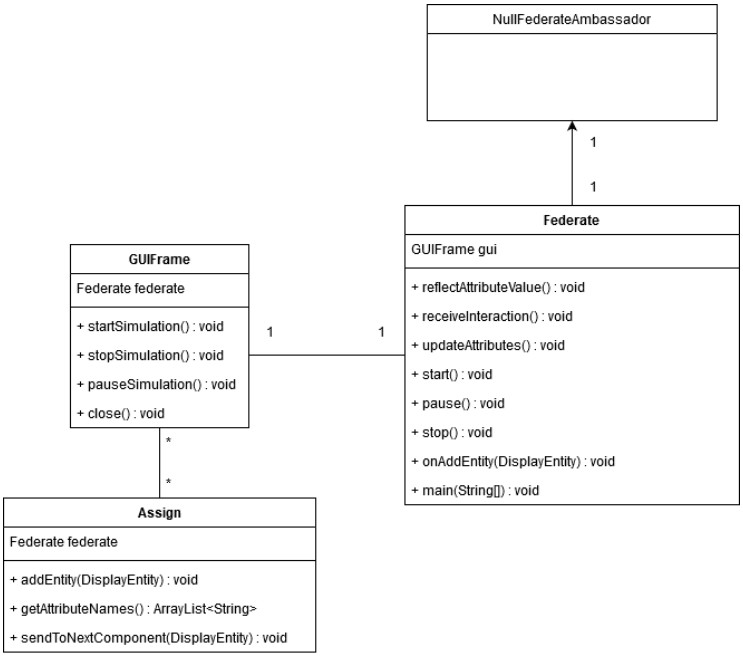


Figure 106 : Diagramme de classes JaamSim - HLA

Durant l'exécution du processus, à chaque fois qu'une entité traverse un composant « *Assign* », la méthode « *addEntity()* » est exécutée. Cette méthode récupère les attributs du composant, effectue les calculs d'attributs définis par l'utilisateur, et passe l'entité manipulée au composant suivant. C'est à cet endroit (voir Tableau 7) que l'on peut extraire les attributs (nos indicateurs de performance) pour les publier au RTI.

```
Class Assign{
	//[…]
	public void addEntity(DisplayEntity ent){ //on entre dans le composant
	Assign
		super.addEntity(ent);

		//on récupère la liste des attributs déclarés
		ArrayList<String> atts = this.getAttributeNames();
		//si parmis ces attributs se trouve « waitRTIOrder »
		si(atts.size() > 0 ET atts.get(0).equals("waitRTIOrder")) alors
			//on exécute la fonction onAddEntity() de la classe fédéré
			Federate.instance.onAddEntity(this);

		this.sendToNextComponent(ent);
	}
}
```

Tableau 7 : Récupération des attributs depuis un composant Assign

Si parmi les attributs de l'Assign exécuté se trouve un attribut nommé « waitRTIOrder », JaamSim exécute la méthode « *onAddEntity()* » de la classe « Federate » (voir Figure 106) en passant en paramètre l'ensemble des attributs ainsi que leurs valeurs. Depuis la classe « Federate », on récupère les valeurs des indicateurs de performances (LEADTIME, WIP, SIMTIME) et on les met à jour via la méthode *updateAttributeValues()*.

L'ensemble de ces classes et méthodes permettent donc d'extraire des variables internes à JaamSim pour les publier dans la fédération HLA. Dans la section suivante, nous étudierons l'interconnexion entre JaamSim et Papyrus.

3.1.6.Connexion JaamSim et Papyrus

Un certain nombre d'étapes sont nécessaires afin de permettre une communication entre JaamSim et Papyrus. La Figure 107 illustre l'ensemble des étapes effectuées entre les deux outils afin d'initialiser une fédération liant les différents composants.

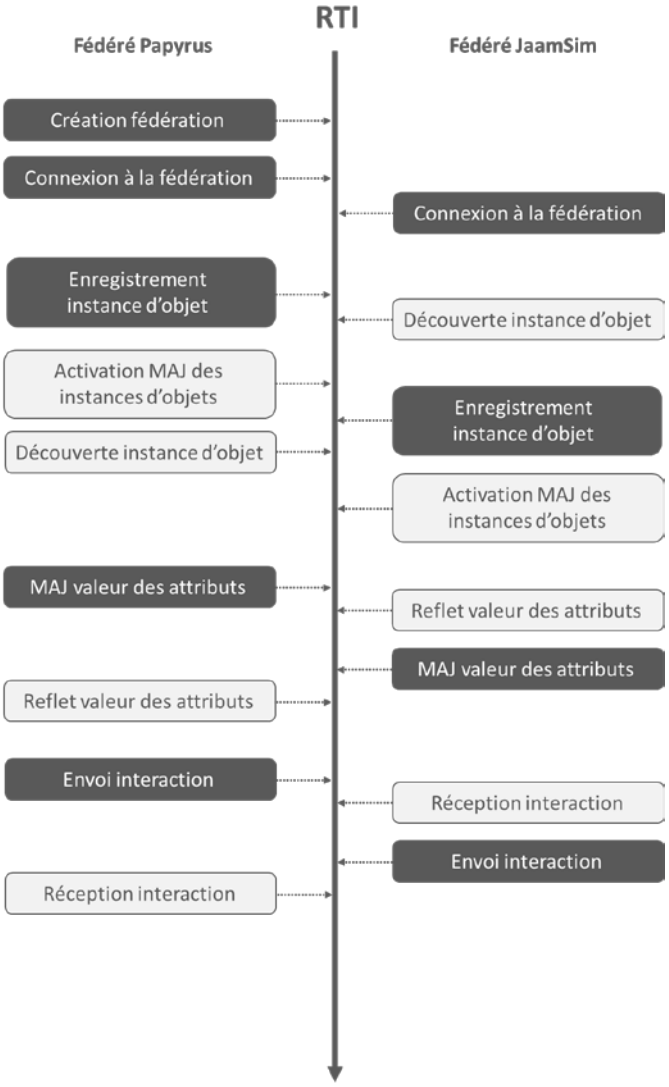


Figure 107 : Implémentation communication HLA entre Papyrus et JaamSim

La première étape concerne la création de la fédération. Papyrus crée la fédération, importe le fichier FOM contenant toutes les informations sur les données à échanger entre les fédérés (objets – attributs et interactions – paramètres permettant la communication de la couche décideur/exécutant HLA, les paramètres de définition de scénario, et l'exportation des résultats de simulation JaamSim). Puis, le fédéré Papyrus et les fédéré JaamSim rejoignent la fédération.
La deuxième étape est l'enregistrement de l'instance d'objets au travers de la méthode *registerObjectInstance()* pour notifier au RTI qu'une nouvelle instance

d'objet a été créée. Lorsque l'instance de l'objet a été créée, les fédérés connectés découvrent l'objet et activent les mises à jour de l'instance d'objet par le biais du callback RTI avec les méthodes *discoverObjectInstance()* et *turnUpdateOnForobjectInstance()*. Cette action est effectuée par chaque fédéré de la simulation. Puis, chacun des fédérés, publie les objets et attributs dont il a besoin pour fonctionner, et s'abonne aux objets et attributs requis auprès du fédéré décideur : Papyrus. Les méthodes *updateAttributeValues()* et *reflectAttributeValues()* sont utilisées pour échanger des objets et attributs entre fédérés.
Pour ce qui est de la publication et de la souscription aux interactions/paramètres, deux étapes sont nécessaires avant de lancer les processus d'envoi/réception des données : obtenir le gestionnaire de classe d'interaction, puis le gestionnaire de paramètres. Les méthodes à utiliser pour envoyer et recevoir des interactions/paramètres sont *sendInteraction()* et *receiveInteraction()*.

Une fois ces étapes d'initialisation effectuées, les fédérés JaamSim se mettent en attente de recevoir deux informations :

- Les paramètres d'entrée nécessaires à l'exécution de la simulation : ces informations sont contenues dans un fichier json côté Papyrus, qui sera transmis au fédéré concerné par la mise à jour d'attributs d'un objet HLA. A la notification de ce changement, JaamSim est en possession de ses paramètres d'entrée, il est en attente d'un ordre d'exécution.
- Une interaction d'ordre d'exécution : Une fois la simulation configurée, Papyrus émet une interaction destinée au composant JaamSim. A la réception de cette interaction, la simulation commence.

Jusqu'à présent nous avons détaillé le fonctionnement et l'interconnexion de JaamSim et Papyrus. La section suivante décrit le fonctionnement du tableau de bord distribué.

3.1.7.Outil de visualisation

Afin de permettre une visualisation de l'évolution des indicateurs de performance JaamSim en temps réel, nous avons développé un tableau de bord distribué. Basé sur *JfreeChart*, cet outil est lancé lors du démarrage de la fédération et de la connexion des fédérés. Il intercepte les indicateurs de performance envoyés depuis les fédérés JaamSim et les affiche en temps réel sur des graphiques qui aident l'utilisateur à suivre les résultats au cours de la simulation. Les données affichées par le fédéré tableau de bord sont les résultats envoyés des fédérés JaamSim à destination de Papyrus par le biais des mécanismes de publication / souscription de la norme HLA. Lorsqu'un des fédérés appelle la méthode *updateAttributeValues()* pour mettre à jour ses attributs, la méthode *reflectAttributeValues()* dans le fédéré d'affichage retourne les valeurs mises à jour et les affiche en temps réel sur le graphe.
Depuis le tableau de bord, nous pouvons visualiser l'évolution des temps de livraison des matières premières, la quantité de produis « encours », et les délais de mise en œuvre de production. De plus, en fin de simulation, l'outil interroge le FMU d'aide à la gestion des risques environnementaux afin d'obtenir un résumé

des impacts météorologiques sur la simulation. La communication entre le fédéré tableau de bord et le FMU est basée sur la contribution du chapitre 5.

Chacun des composants détaillés dans les sous-sections précédentes sont regroupés dans une seule application détaillée dans la partie suivante.

3.2.Orchestration globale

La Figure 108 introduit l'architecture générale dirigée par Papyrus orchestrant les composants distribués :

- Le fichier Excel de gestion des risques industriels qui contient l'ensemble de données relatives aux aléas inhérents au contexte de l'entreprise
- Le FMU de gestion des risques environnementaux qui contient les outils nécessaires à l'extraction d'informations météorologiques pour notre contexte industriel
- Un RTI Pitch permettant l'interconnexion et la communication de fédérés
- Une instance JaamSim compatible HLA et modifiée pour notre cas d'étude simulant la phase d'approvisionnement de matériaux
- Une instance JaamSim compatible HLA et personnalisée simulant le déploiement de l'usine mobile
- Un outil de visualisation de données en temps réel compatible HLA et FMI
- Le processus d'orchestration de la simulation distribuée Papyrus

Papyrus et JaamSim sont configurés comme des fédérés distincts pouvant fonctionner sur des machines différentes. La communication entre ces éléments est basée sur les mécanismes de publication/souscription de la norme HLA. Ainsi, les objets/attributs et interactions/paramètres sont nécessaires afin de configurer, contrôler et exécuter correctement le modèle de simulation tel qu'il est présenté sur la Figure 108.

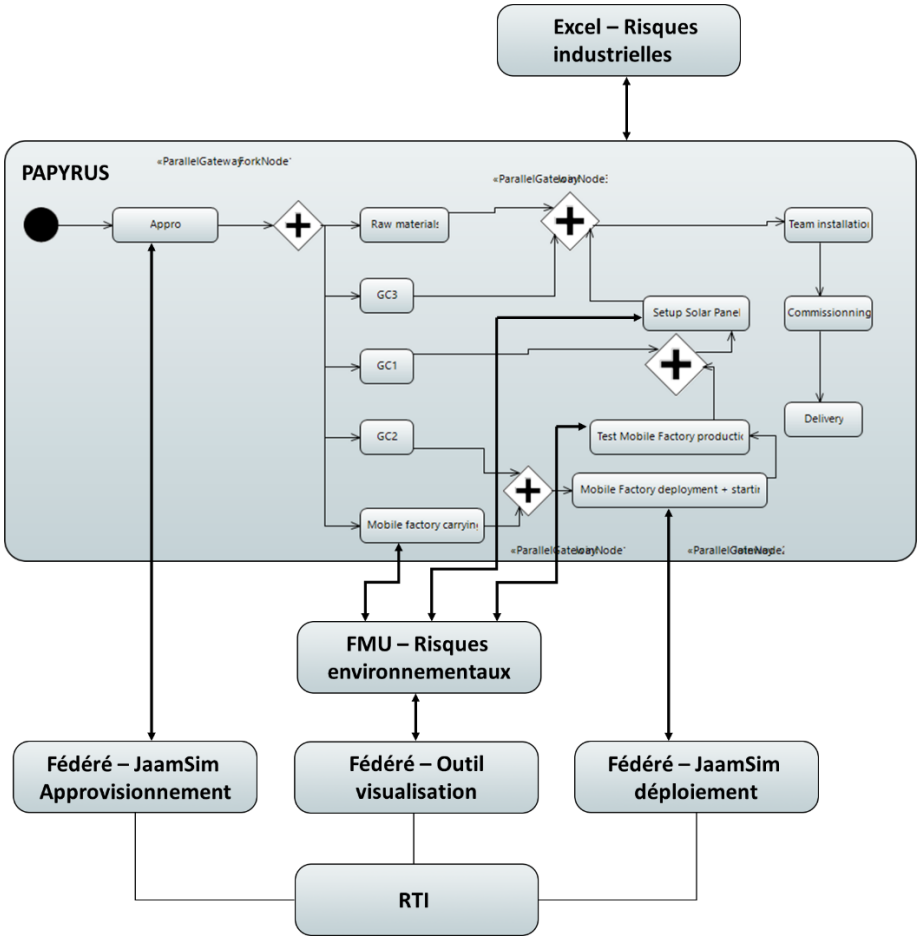


Figure 108 : Orchestration globale

Papyrus étant l'orchestrateur, il doit être exécuté en premier. Comme nous l'avons vu plus tôt, l'ensemble des extensions apportées à l'outil lui octroient un certain nombre de fonctionnalités supplémentaires.

La couche HLA (chapitre 5) permet à la fois une communication avec les membres d'une fédération, et à la fois l'implémentation d'un mécanisme de décideur/exécutant entre les fédérés et Papyrus. Deux exécutions de JaamSim déportées ont lieux à deux moments différents de la simulation Papyrus : durant la tâche *Appro*, et la tâche *Mobile factory deployment*. Ces deux tâches sont monitorées par l'outil de visualisation. Connecté à la fédération en tant que fédéré autonome, il aide l'utilisateur à suivre le processus de production de l'entreprise en temps réel avec la réception et l'affichage des données envoyées par les fédérés.

La couche de gestion des risques industriels (chapitre 3) permet une communication entre le moteur de simulation de Papyrus (Moka) et des outils extérieurs (Excel ou base de données). L'ensemble de ces lois des tables Excel dépend de facteurs semi-aléatoires (reproductibles) et du temps de la simulation afin de perturber l'exécution. Ces perturbations auront effet dans différentes tâches du processus. Une fois la simulation terminée, l'utilisateur pourra consulter les effets qui auront eu lieu au cours de la simulation en consultant le fichier Excel.

La couche FMI (chapitre 4) permet à Papyrus de lancer des modules de co-simulation (FMU). Le cas d'étude étant une centrale électrique à concentration de chaleur, les résultats des requêtes sur les données météorologiques auront une influence sur certaines tâches de la simulation définies par l'utilisateur. Dans notre cas, les tâches *Mobile factory carrying*, *Setup Solar Panel*, et *Test Mobile Factory Production*. De plus, le fédéré de visualisation graphique effectue une requête de synthèse au FMU météorologique pour récupérer un résumé des informations transmises par celui-ci durant la simulation.

La section suivante illustre l'architecture globale par un exemple d'application inspirée du domaine d'ALSOLENTECH.

4.Processus de simulation et résultats

L'objectif de ce chapitre est de modéliser l'ensemble du processus de production du projet SolR². A la croisée de plusieurs disciplines, ALSOLENTECH exprime le besoin d'un outil capable de regrouper des standards de simulation, de simplifier la modélisation de processus en externalisant les risques, et de connecter des outils externes de son environnement. Dans cette section, nous regroupons l'ensemble des contributions énoncées précédemment afin de former une application globale permettant l'unification d'outils et de technologies. Papyrus, l'outil d'orchestration interconnecte des blocs de fonctionnalité, chacun représentant une partie du processus de production.

La Figure 109 représente le processus de mise en place d'un champ solaire. Il regroupe un ensemble de tâches faisant référence à des outils externes.

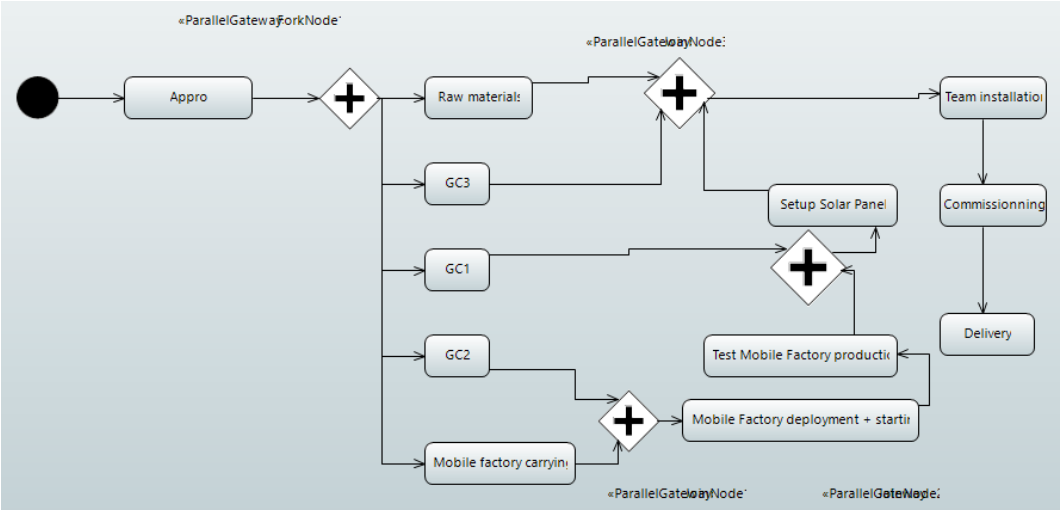


Figure 109 : Papyrus : Orchestration globale

La première tâche consiste en l'approvisionnement par avion de deux catégories de produits (tâche « Appro »). Cette tâche est soumise aux risques et sera déportée vers une simulation JaamSim. Nous avons donc deux profils appliqués sur cette action UML visible sur la Figure 110 :

- FailureProfile pour connecter la tâche à l'outil de gestion des risques

- HLAProfile pour connecter la tâche à un fédéré HLA.

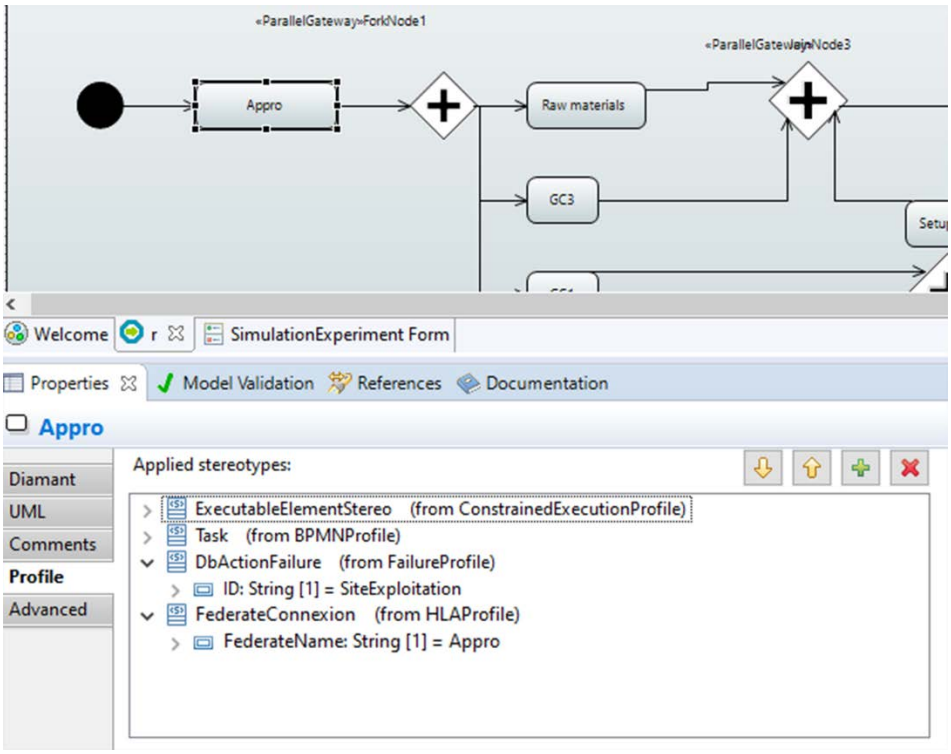


Figure 110 : Profils appliqués à la tâche « Appro »

Dans la Figure 110, on transmet l'identifiant de la tâche au fichier Excel par le biais du champ ID contenu dans le FailureProfile -> SiteExploitation. Cette tâche Papyrus a pour objectif de lancer une simulation JaamSim. On lui applique également le profil HLAprofile, qui pointe vers le nom du fédéré à exécuter -> Appro. Cet approvisionnement étant soumis aux risques, Papyrus interroge l'outil de gestion des risques environnementaux, afin d'influencer les paramètres d'entrée de la simulation JaamSim. Les paramètres sont calculés puis transmis au fédéré JaamSim par la modification de l'Objet HLA de la Figure 111.

Scenario_Appro			ps
Federate_name	HLAunicodeString	ps	ro
SimTime	HLAfloat64Time	ps	ro
N_HPriority	HLAinteger64Time	ps	ro
N_LPriority	HLAinteger64Time	ps	ro
F_HPriority	HLAunicodeString	ps	ro
F_LPriority	HLAunicodeString	ps	ro
D_Loading	HLAunicodeString	ps	ro
D_TR	HLAunicodeString	ps	ro
D_VOL	HLAunicodeString	ps	ro
D_TR2	HLAunicodeString	ps	ro
D_Depal	HLAunicodeString	ps	ro

Figure 111 : Objet HLA : Scénario Approvisionnement

Ici, l'ensemble des paramètres sont des variables près-configurées par l'utilisateur. Ce sont les paramètres nominaux de la simulation qui ne prennent pas en compte le risque. L'ensemble de ces valeurs peut être modifié par l'utilisateur en

configurant l'outil de gestion des risques. Dans un second temps, Papyrus envoie une interaction d'ordre d'exécution au fédéré qui exécute la simulation visible sur la Figure 112.

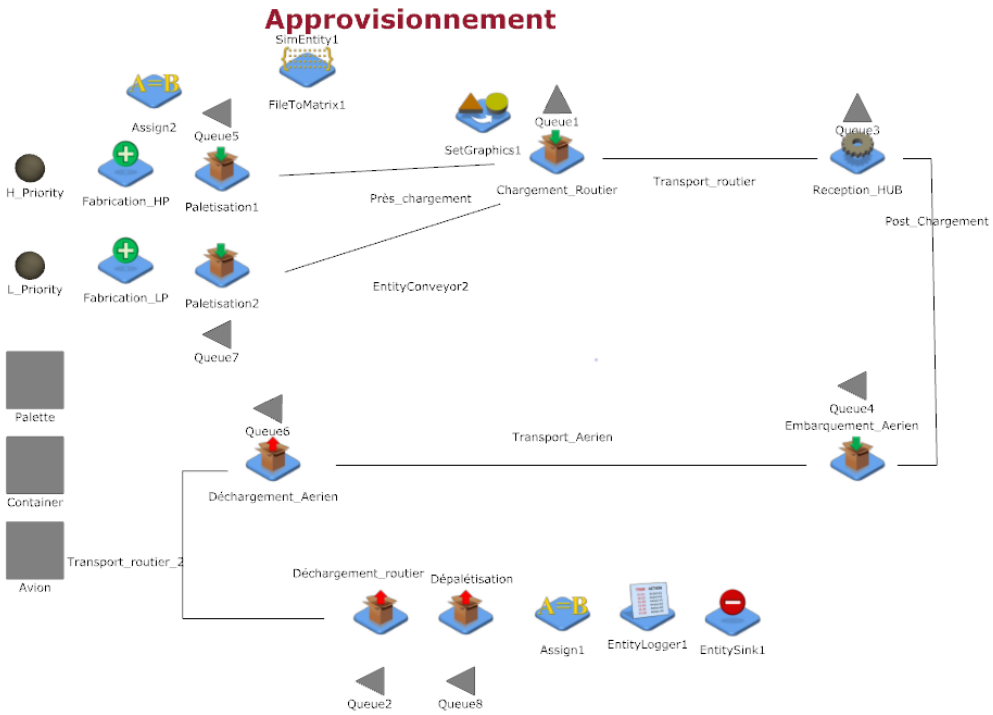


Figure 112 : Simulation JaamSim : Approvisionnement SolR²

La simulation d'approvisionnement est un processus JaamSim générant deux types d'entités (H_Priority et L_Priority). Leur nombre et leur cadence de production sont des variables d'entrée. De plus, les différents temps d'exécution relatifs aux transports, transbordements, chargements, déchargements sont également des paramètres d'entrée pouvant être modifiés par les outils de gestion du risque. Au cours de cette simulation, les différents indicateurs de performance sont relevés, et mis à jour au près du RTI à chaque traversée du block « Assign1 » de la Figure 112. La mise à jour de ces paramètres (Figure 113) est effectuée avec la fonction *updateAttributeValues()* de l'ambassadeur RTI.

KPI		ps	
FederateName	HLAunicodeString	ps	ro
SimTime	HLAfloat64Time	ps	ro
LEADTIME	HLAinteger64Time	ps	ro
WIP	HLAinteger64Time	ps	ro

Figure 113 : Objet HLA : KPI

De son côté, l'outil de visualisation récupère les données émises par le fédéré afin d'en afficher les résultats à l'utilisateur. La Figure 114 représente l'évolution au cours du temps des délais de livraison des matières premières sans prise en compte des risques. Cette valeur détermine le temps qui s'écoule entre le moment ou un produit est palettisé (voir étapes « palettisation 1 et 2 » de la Figure 112) et le moment où il est déchargé du camion (« voir dé-palettisation »). Dans notre cas, on constate que les matériaux peuvent être présents sur le chantier de construction dans des délais oscillant entre 26 et 33 heures.

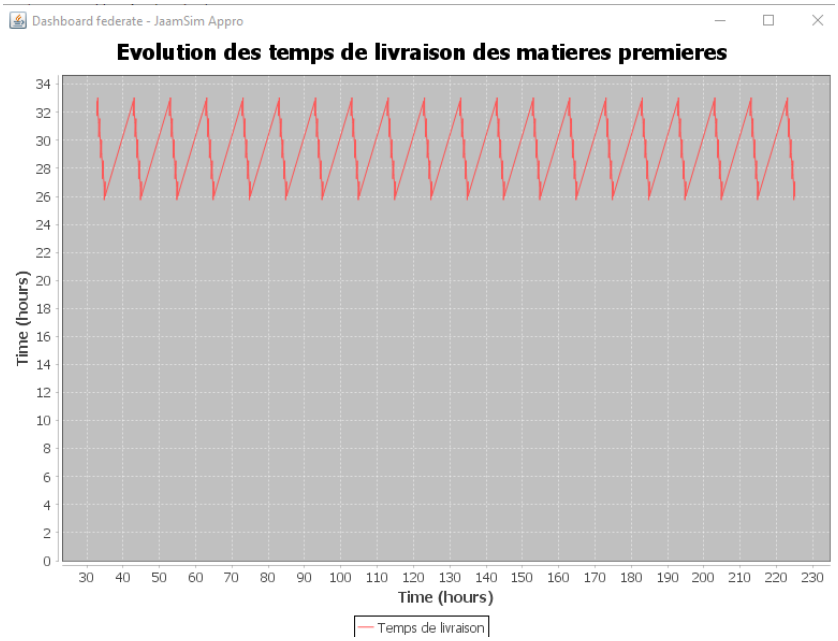


Figure 114 : Evolution des délais de livraison sans prise en compte des risques

Les étapes suivantes du processus Papyrus sont exécutées en parallèle :

- L'approvisionnement de matériel et matières premières pour la mise en place du champ solaire (Raw materials, GC1, GC2, GC3) sont des tâches simulées par Papyrus et perturbées par la base de données de gestion des risques industriels.
- La livraison de l'usine mobile est simulée par Papyrus et perturbée par l'outil de gestion des risques environnementaux (FMU de la Figure 115). Ce composant externe à Papyrus est un système composé de différents états effectuant des requêtes météorologiques afin de perturber le déroulement d'une tâche. Dans notre cas, c'est la première fois qu'une tâche Papyrus fait appel au FMU. Il est donc à son état initial : requêtes sur le taux d'humidité (précipitations) d'une zone géographique pour un temps décrit dans le fichier de configuration de la simulation.

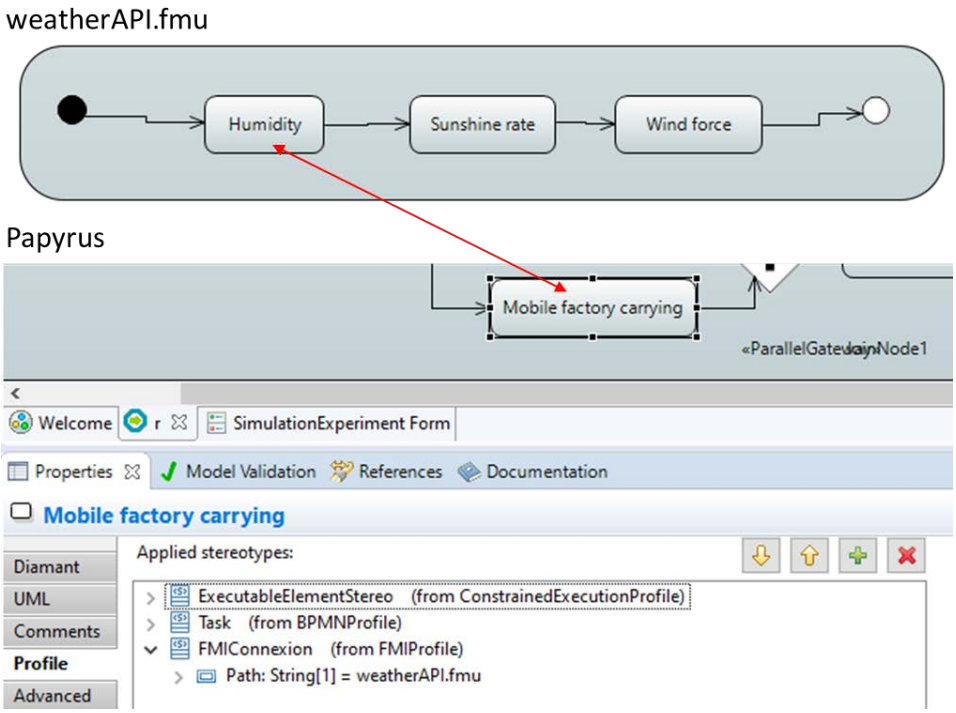


Figure 115 : Process FMU météorologique

Après l'exécution de sa première tâche, le FMU retourne la valeur de dégradation calculée en fonction des moyennes d'humidité et de précipitation de la zone définie. Dans un second temps, Papyrus avancera le FMU à son état suivant : taux d'ensoleillement destiné à la prochaine utilisation du FMU.

La tâche suivante du modèle orchestrateur concerne le déploiement de la centrale. Ici, Papyrus délocalise une nouvelle fois la complexité de simulation en passant par un modèle JaamSim soumis aux risques industriels. La tâche de la Figure 116 est donc étendue par le profil *HLAProfile* (qui indique le nom du fédéré JaamSim à exécuter) et le profil *FailureProfile* (qui introduit des aléas dans les entrées du simulateur).

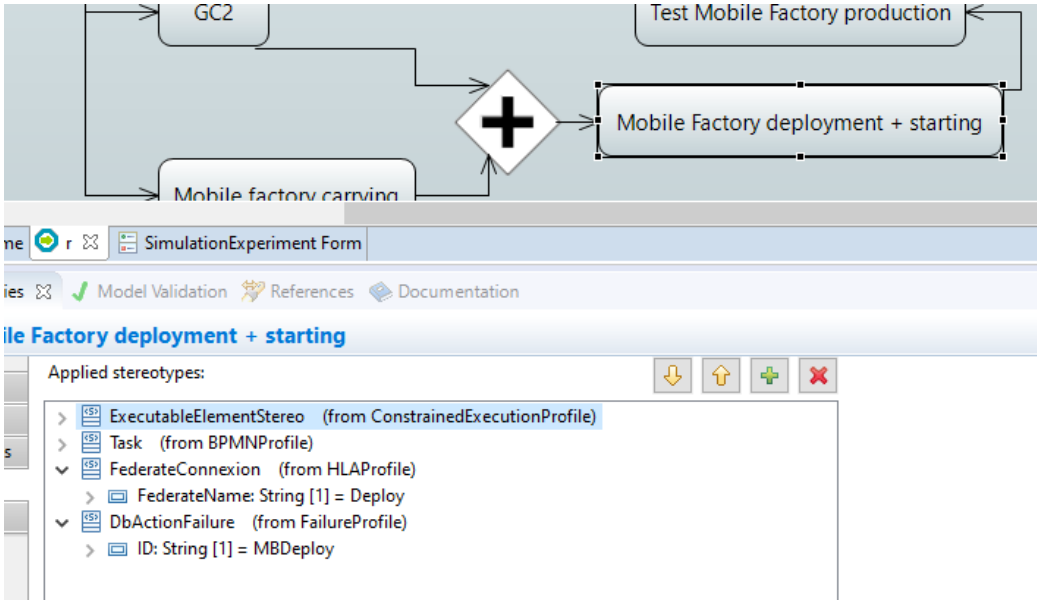


Figure 116 : Profils appliqués au déploiement de la centrale

Le déploiement de la centrale est une tâche simulée par JaamSim. Il est question ici de construire et de déployer les réflecteurs solaires directement sur le site de production. Son fonctionnement (illustré par la Figure 117) est une succession d'étapes transformant des tôles d'acier en des miroirs posés sur portiques motorisés.

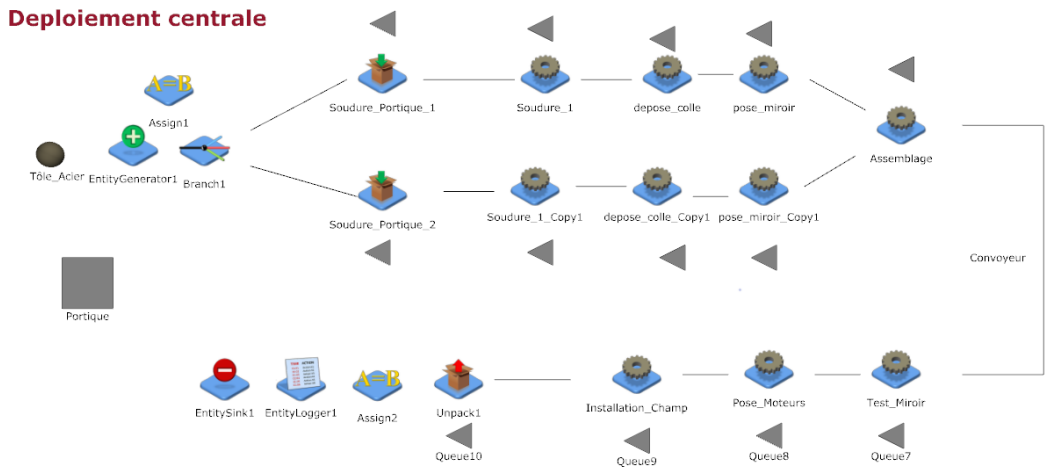
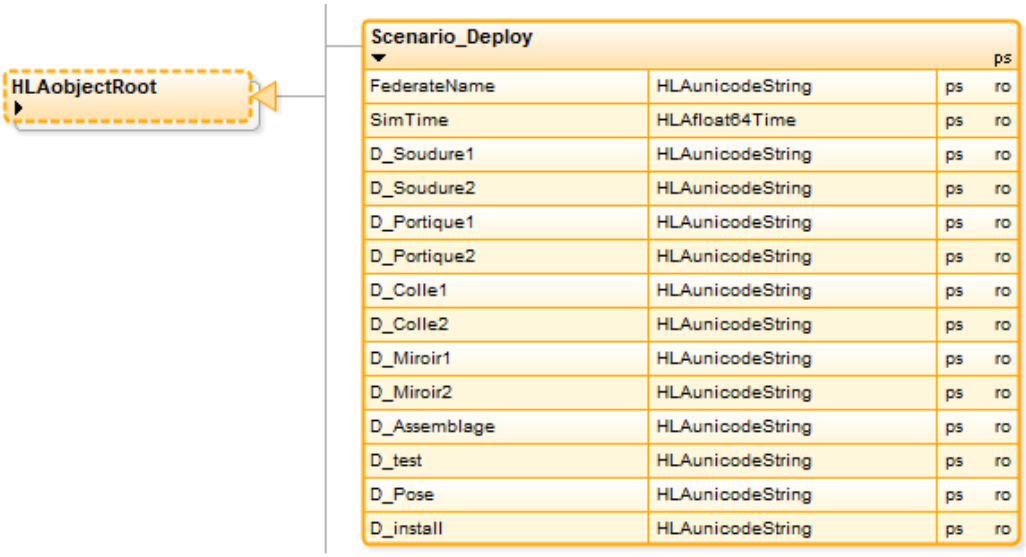


Figure 117 : Simulation JaamSim : Déploiement centrale

Comme pour la précédente simulation JaamSim, l'ensemble des délais de production illustrés dans la Figure 117 sont des paramètres d'entrée de la simulation. Définis par défaut dans le fichier de configuration Papyrus, ils sont accessibles en modification par l'outil de gestion des risques industriel. Au moment de l'exécution de la tâche, Papyrus transmet les paramètres par le biais d'un Objet HLA : tel que dans la Figure 118, puis envoie l'interaction d'ordre d'exécution.



FederateName	HLAunicodeString	ps	ro
SimTime	HLAfloat64Time	ps	ro
D_Soudure1	HLAunicodeString	ps	ro
D_Soudure2	HLAunicodeString	ps	ro
D_Portique1	HLAunicodeString	ps	ro
D_Portique2	HLAunicodeString	ps	ro
D_Colle1	HLAunicodeString	ps	ro
D_Colle2	HLAunicodeString	ps	ro
D_Miroir1	HLAunicodeString	ps	ro
D_Miroir2	HLAunicodeString	ps	ro
D_Assemblage	HLAunicodeString	ps	ro
D_test	HLAunicodeString	ps	ro
D_Pose	HLAunicodeString	ps	ro
D_install	HLAunicodeString	ps	ro

Figure 118 : Objet HLA : Paramètres de simulation JaamSim : Déploiement centrale

Les indicateurs de performance liés à cette simulation sont :

- L'évolution des délais de mise en œuvre (LEADTIME)
- L'évolution des encours dans le temps (WIP)

Dans un scénario ou la gestion des risques n'influence pas la simulation JaamSim, on peut observer sur le fédéré de visualisation les résultats visibles sur la Figure 119.

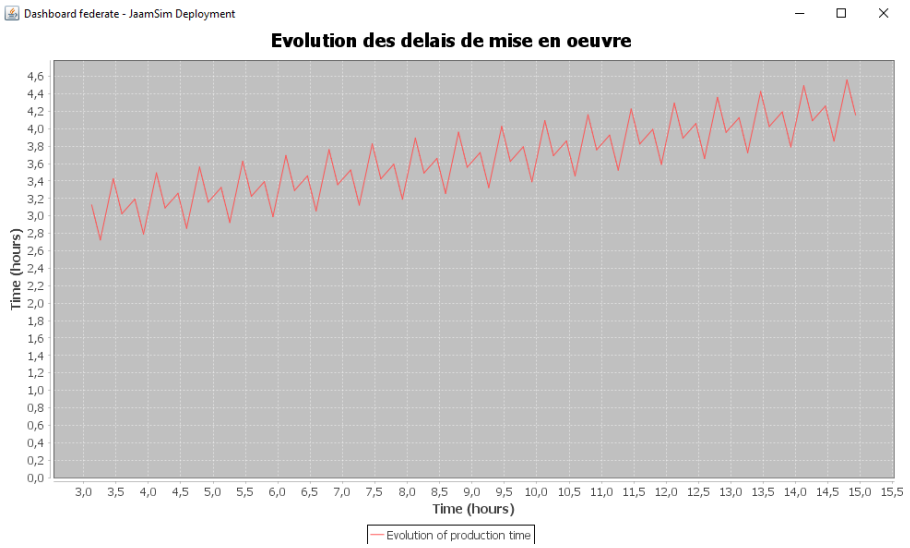


Figure 119 : Délai de mise en œuvre dans un mode de fonctionnement nominal

L'évolution des délais de mise en œuvre se calcule à partir du temps qui sépare l'entrée d'un produit dans le processus de production, de sa sortie. Dans la simulation JaamSim, on imprime sur chacun des produits la date à laquelle il est inséré dans la simulation (au niveau de l'Assign1 de la Figure 117). A la sortie du processus (Assign2), on soustrait la date courante à la date contenue dans le produit. Ici, on observe que le délai de production d'un produit a une tendance à la hausse. L'ensemble des produits mettent 15h à être fabriqués.

La Figure 120 représente le deuxième indicateur à notre disposition Il représente l'évolution de la quantité d'encours dans le temps. A chaque passage dans le

module Assign2 de la simulation JaamSim, nous récupérons la somme des produits en file d'attente sur tout le processus de production. Cette valeur indique le nombre de produits en attente de traitement.

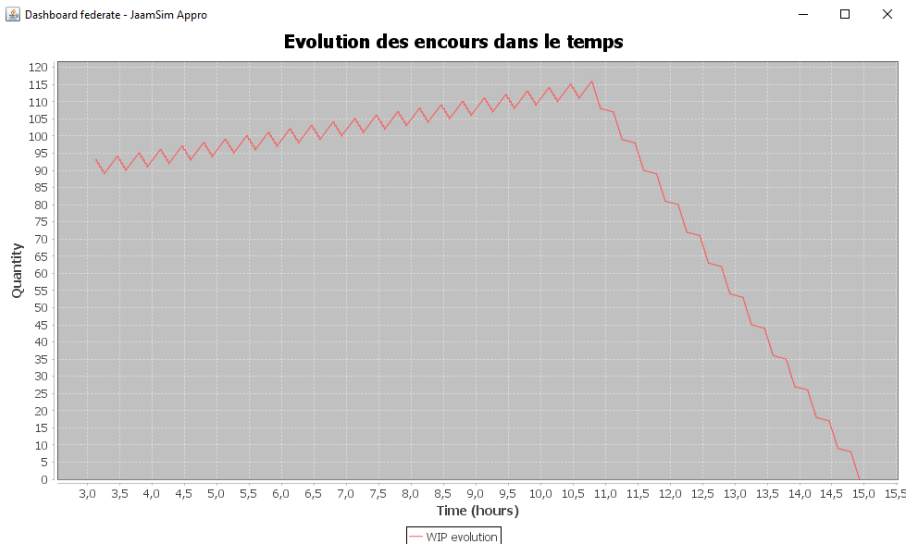


Figure 120 : Evolution de la quantité d'encours dans le temps

Le résultat de simulation que nous observons est le fruit d'une simulation JaamSim sans prise en compte des modules générant les risques. On remarque que la simulation met 15h à se terminer, avec une valeur d'encours crête à 115 produits. La baisse d'encours à partir de 11h de simulation représente le moment ou toutes les matières premières sont injectées dans l'usine mobile. La simulation se terminant 4 heures plus tard, cela confirme le graphique précédent : en fin de simulation, un produit met 4 heures à être traité entièrement par l'usine.
L'ensemble des étapes suivantes de la simulation Papyrus est une succession d'appels aux outils de gestion des risques industriels et environnementaux. Nous pouvons donc classer l'ensemble des tâches Papyrus en fonction des profils qui leur sont appliqués et des identifiants qui leur sont transmis. Ceci est résumé dans le Tableau 8.

	Profiles UML		
Tâches Papyrus	**FailureProfile**	**HLAProfile**	**FMIProfile**
Appro	SiteExploitation	Appro	
Raw materials	raw		
GC1	GC1		
GC2	GC2		
GC3	GC3		
Mobile Factory carrying			weatherAPI.fmu
Mobile Factory deployment	MBDeploy	Deploy	
Test Mobile Factory Production			weatherAPI.fmu
Setup Solar Panels			weatherAPI.fmu
Team installation	Installation EQUIP		
Commissionning	Commissionning		
Delivery	Livraison		

Tableau 8 : Récapitulatif des profils UML appliqués aux tâches Papyrus

Papyrus est un outil de modélisation et de simulation pouvant fonctionner de manière autonome. Par défaut, l'orchestration de la Figure 109 peut être exécutée sans extension de gestion des risques. Chacune des tâches du modèle (à l'exception des tâches faisant référence aux simulations JaamSim) possède une durée nominale définie par l'utilisateur dans le Tableau 9. L'ensemble de ces durées est déterminé par les ingénieurs qui estiment le temps moyen de chacune des étapes.

Durée	Tâche Papyrus
91h	Raw materials
43h	GC3
139h	GC1
205h	GC2
21h	Mobile Factory Carrying
37h	Test Mobile Factory Production
10h	Setup Solar Panels
8h	Team Installation
15h	Commissionning
3h	Delivery

Tableau 9 : Durées nominales des tâches Papyrus

Les deux tâches Papyrus manquantes sur le Tableau 9 (Appro et Mobile Factory deployment) sont les deux tâches qui réfèrent aux simulations JaamSim. En effet, le temps de leur exécution n'est pas inscrit sur le modèle UML. Il sera déterminé au cours de la simulation par l'extension Moka gérant la communication HLA. Une fois la modélisation terminée, l'utilisateur peut lancer une simulation Papyrus en faisant le choix de ne pas inclure la gestion des risques à la modélisation. On obtient donc le résultat sur la Figure 121. La simulation débute le 03/07/2018 à 8h00m00s et se termine le 24/07/2018 à 8h00m00s. L'outil de visualisation des résultats JaamSim affiche les courbes visibles sur les Figure 114, Figure 119, Figure 120, et ne détecte pas la présence du FMU de gestion des risques environnementaux.

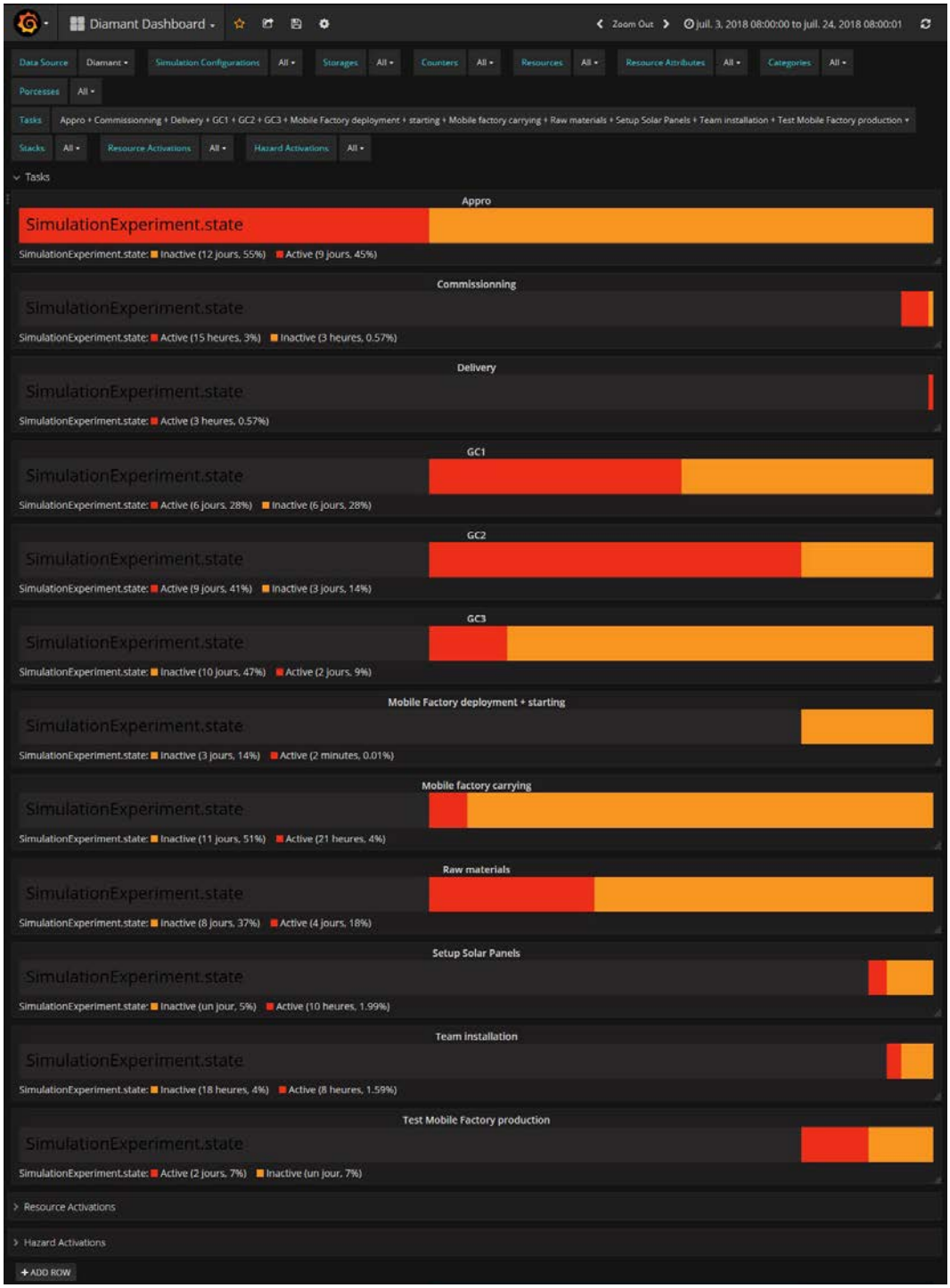


Figure 121 : Résultats simulation Papyrus sans prise en compte des risques

Dans un second temps, on conserve le même jeu de données d'entrée et on connecte les deux outils de gestion des risques à la Simulation Papyrus. Ces deux outils ajouteront des aléas maitrisés durant le déroulement de la simulation. Le résultat de simulation Papyrus est visible sur la Figure 122. On constate que la simulation débute toujours le 03/07/2018 à 8h00m00s mais se termine le 29/07/2018 à 09h00m00s.

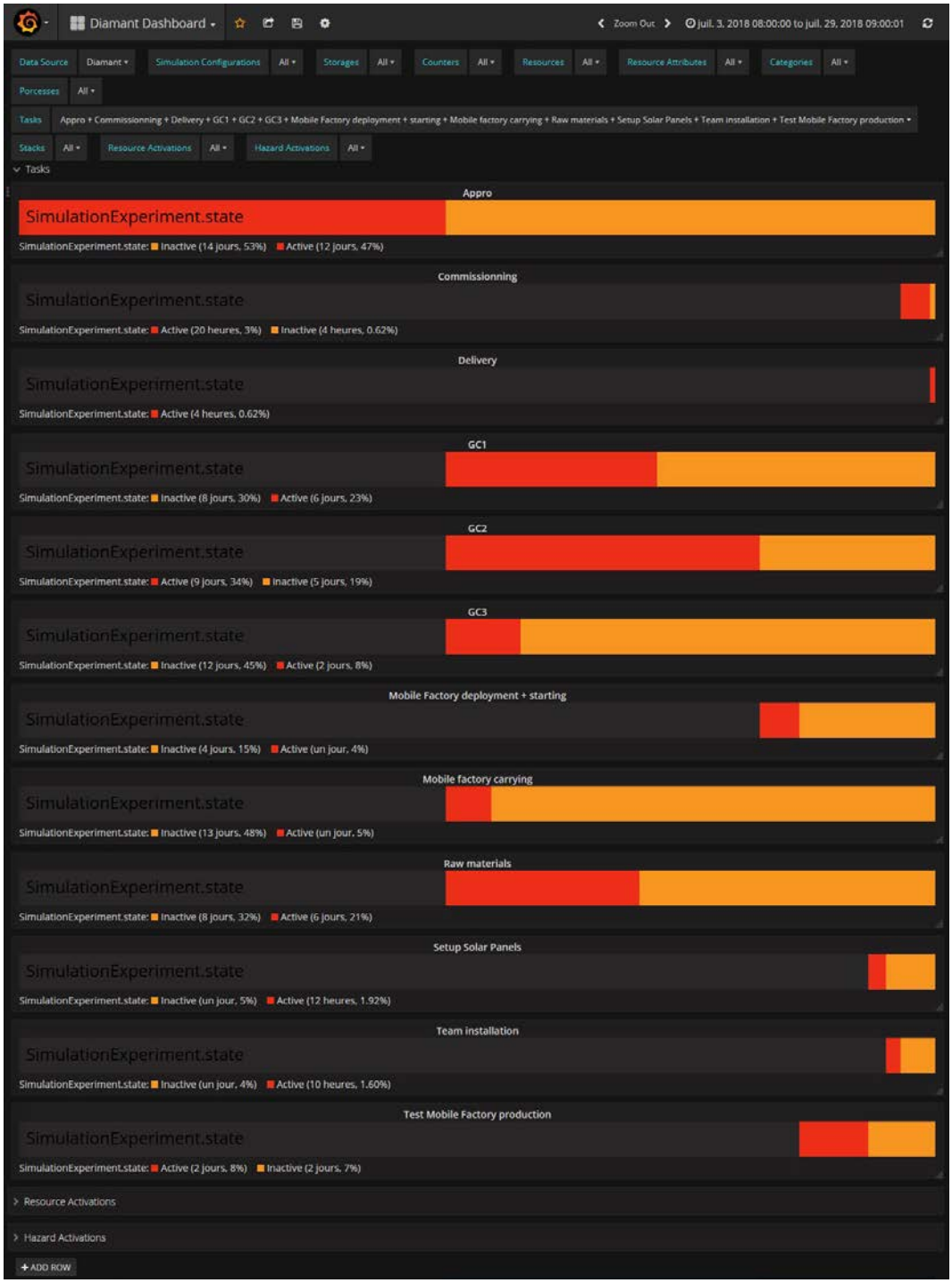


Figure 122 : Résultats simulation Papyrus avec prise en compte des risques

A la consultation du fichier Excel visible Tableau 10, on peut obtenir les détails des dégradations temporelles de chacune des tâches Papyrus, ainsi que l'heure simulée au moment où ces tâches ont été exécutées.

Tâche Papyrus	Durée tâche Papyrus nominal	Durée ajoutée par Excel/FMU	Date simulation
Raw materials	91h	41h	15/7/18 11:01:00
SiteExploitation.Apro.D_VOL	x	7h	3/7/18 8:00:00
SiteExploitation.Apro.D_Depal	x	37m	3/7/18 8:00:00
GC3	43h	8h	15/7/18 11:01:00
GC1	139h	5h	15/7/18 11:01:00
GC2	205h	9h	15/7/18 11:01:00
Mobile Factory Carrying	21h	FMU (10h)	x
MBDeploy.Deploy.D_Install	x	20m	24/7/18 9:02:00
MBDeploy.Deploy.D_Pose	x	10m	24/7/18 9:02:00
MBDeploy.Deploy.D_test	x	2m	24/7/18 9:02:00
Test Mobile Factory Production	37h	FMU (10h)	x
Setup Solar Panels	10h	FMU (2h)	x
Team Installation	8h	2h	27/7/18 23:05:00
Commissionning	15h	5h	28/7/18 9:06:00
Delivery	3h	1h	29/7/18 5:07:00

Tableau 10 : Risques générés par Excel durant la simulation

Le fédéré de visualisation de données indique en fin de simulation les valeurs de dégradation temporelle injectées par le FMU environnemental. Ces valeurs (inscrites en vert sur le Tableau 10) ont influencé les trois étapes Mobile *Factory Carrying*, *Test Mobile Factory Production*, et *Setup Solar Panels*. De plus, l'outil permet de visualiser l'évolution des indicateurs JaamSim déterminés par l'utilisateur en Figure 123 pour la simulation d'approvisionnement et Figure 124 pour le déploiement de la centrale.

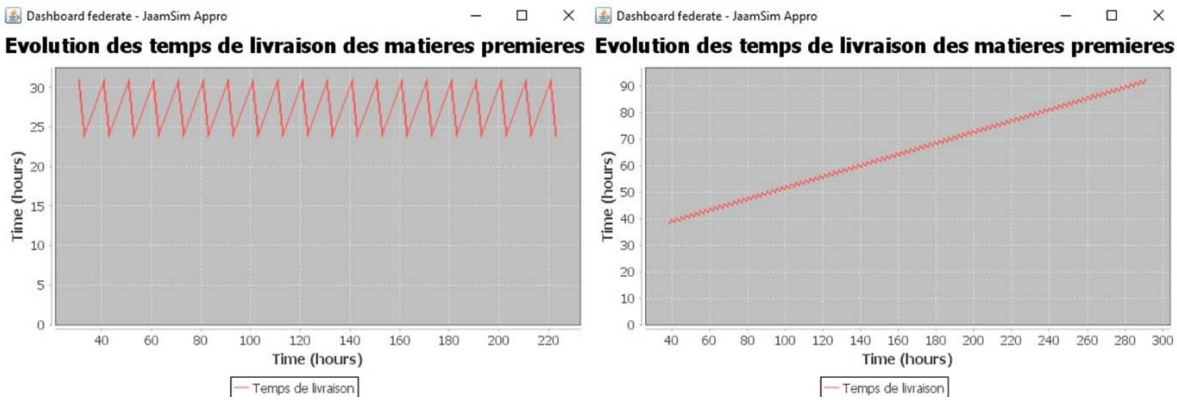


Figure 123 : Comparaison des évolutions de temps de livraison des matières premières avec (figure de droite) et sans gestion des risques (figure de gauche)

Ces deux courbes sont le résultat de l'évolution des temps de livraison des matières premières pour la simulation d'approvisionnement. Elles sont générées à partir du même indicateur de performance (*LEADTIME*), et des mêmes paramètres de simulation. La courbe de gauche est le résultat d'une simulation sans gestion des risques et aléas. On observe que les temps d'acheminement des matières oscillent entre 24 et 31 heures. La courbe de droite est le résultat de la même simulation, avec la gestion des risques et aléas. Selon le Tableau 10, deux paramètres d'entrée à JaamSim ont été affectés lors de l'exécution de la

simulation. *D_VOL* et *D_Depal* ont reçu une augmentation de leur durée respective de 7 heures et 37 minutes. Le paramètre *D_VOL* affecte la duré du vol acheminant les matériaux livrés. Pour une durée nominale de 3 heures, la simulation du transport a donc durée 10 heures. C'est la raison pour laquelle la courbe de droite de la Figure 123 débute à 40 heures et non 30 heures comme prévu initialement.
Le paramètre *D_Depal* correspond au temps nécessaire à la dé-paléttisation de chacun des produits transportés par avion. Pour une durée nominale de 1minute, un évènement a dégradé l'efficacité de cette étape à 38 minutes. Ce changement de paramètre explique l'augmentation des délais de livraison au cours de la simulation car il crée un goulot d'étranglement lors du déchargement des matériaux. La durée de la tâche passe donc de 220 heures à 390 heures.

Le deuxième fédéré JaamSim (Déploiement de la centrale) est lui aussi affecté par des modifications extérieures. Le Tableau 10 nous indique que 3 paramètres d'entrée ont changé :

- *D_Test* : La durée des tests de vérification de pose des miroirs passe de 20 min/produit à 22 min/produit (+2 minutes par produit).
- *D_Pose* : La durée de pose des moteurs sur support de réflecteur passe de 20 min/produit à 30 (+10 minutes par produit).
- *D_Install* : La durée d'installation d'une rangée passe de 20 min/produits à 40 min/produit (+20 minutes par produit)

Le changement de ces paramètres peut se remarquer sur la Figure 124 : les indicateurs de performance (courbes de droite) illustrent une saturation de marchandises. Les 3 étapes impactées par ces aléas étant successives (voir les étapes *Test_Miror*, *Pose_Moteurs*, et *Installation_Champ* de la Figure 117), l'augmentation des taux de production et du nombre de matériaux encours de fabrication sont symptomatiques d'un goulot d'étranglement.

L'ensemble des tâches Papyrus a donc reçu une dégradation temporelle en fonction de divers facteurs :

- Lois semi-aléatoires décrites en fonction des connaissances de l'entreprise (reproductibles avec un système de génération de graines) ;
- Dates de simulation ;
- Facteurs environnementaux liés projet (localisation, date prévue du chantier, précipitations, ensoleillements, etc.) ;

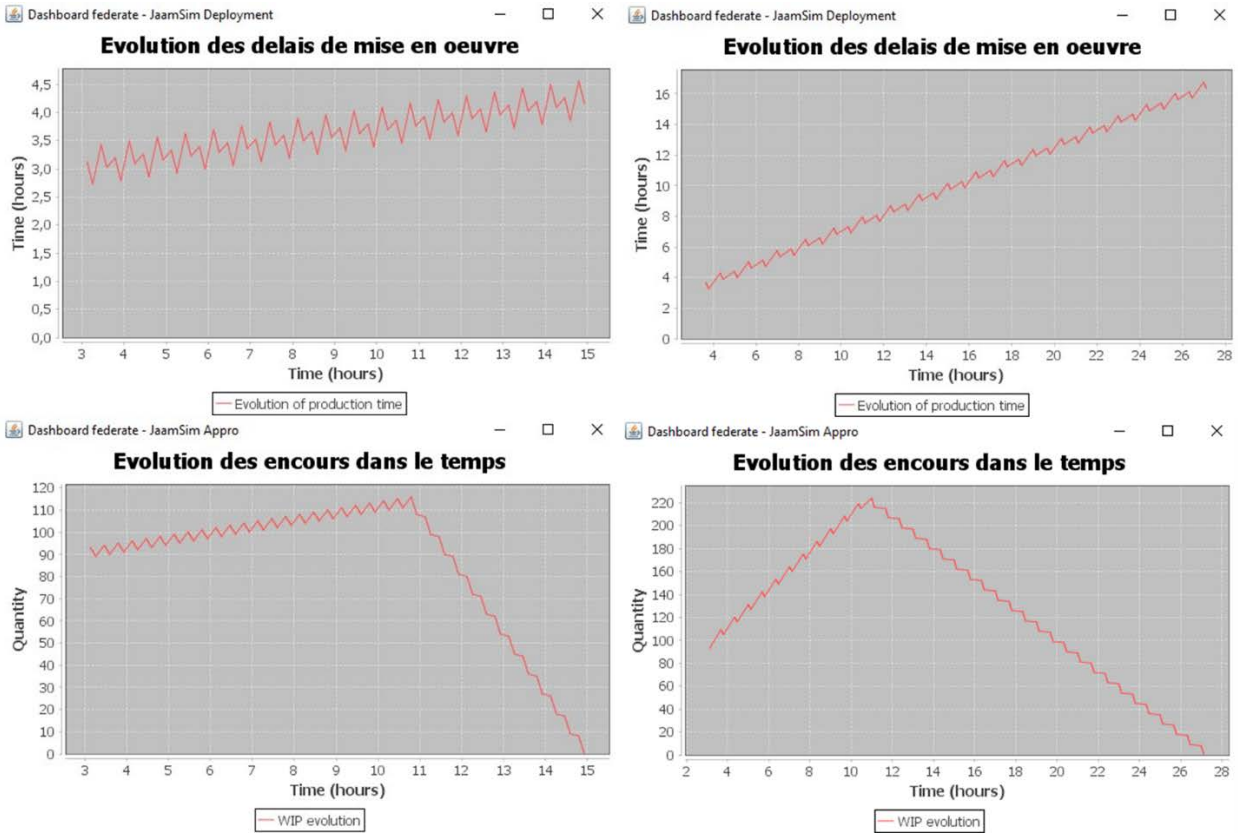


Figure 124 : Comparaison des évolutions de délais de mise en œuvre et d'encours avec gestion des risques (graphiques de droite) et sans gestion des risques (graphiques de gauche)

5.Conclusion

L'outil Papyrus est un moyen efficace pour modéliser et simuler un processus industriel. Nous avons démontré que sa capacité d'extension lui permet de s'ouvrir à d'autres méthodes et technologies. L'effet d'héritage qu'ont les extensions de Papyrus entre elles nous permet d'ajouter des contributions les unes au-dessus des autres. Ainsi, nous sommes en mesure d'ajouter des complexités de modélisation et de simulation sans impacter le modèle de base.

Durant ce chapitre nous avons assemblé les trois contributions majeures de ce manuscrit dans un même outil. La contribution du chapitre 3 nous a permis de connecter une base de données de connaissances des risques industriels à Papyrus, pour que ces informations puissent impacter les résultats de simulation. Le chapitre 4 ouvre l'outil à un standard de communication avec l'extérieur en temps réel (FMI). Nous avons fait la démonstration de l'utilisation d'un composant capable d'extraire des données de l'environnement (au travers du FMU effectuant des requêtes vers un Web Service météorologique). Cet exemple démontre les possibilités d'interconnexion entre un système simulé et son environnement réel. Enfin, la contribution du chapitre 5 démontre les capacités de l'outil à communiquer ou à diriger d'autres simulateurs par le biais d'un standard de simulation distribuée (HLA).

Chapitre 7
Conclusions & Perspectives

Les recherches développées dans ce mémoire de thèse présentent une plateforme collaborative de modélisation et de simulation distribuée fédérant des sous-systèmes de simulation et de gestion des risques. La modélisation et la simulation jouent un rôle central dans le dimensionnement des processus métiers de l'industrie en raison d'une augmentation significative des composants qui y participent. L'accumulation et l'assemblage de composants variés forment un système complexe et induisent une augmentation proportionnelle des risques inhérents aux projets. De plus, un projet étant l'association de plusieurs sous-systèmes variés, les gestionnaires doivent être en mesure d'effectuer des simulations intégrant des composants hétérogènes, indépendants, et distribués. Lorsqu'un modèle est complexe ou nécessite des traitements distribués, l'exécution de ce dernier doit être divisée et répartie entre plusieurs processeurs ou machines. Cette approche de modélisation et simulation doit à la fois prendre en compte la notion de distribution de la complexité, mais aussi la gestion des risques afin de sécuriser la mise en œuvre et l'exécution du système proposé.

Dans la littérature, les outils traditionnels de simulation de processus industriels prennent en compte la gestion des risques et des aléas lors des phases de modélisation et de définition des scénarios. Ces tâches augmentent la complexité de représentation d'un système, et saturent en informations les données d'entrée d'une simulation. De plus, le pilotage d'un processus industriel complexe implique la communication avec des sous-systèmes variés, hétérogènes et distribués. On peut donc identifier un besoin d'orchestration de composants hétérogènes dirigés par un processus métier. Au regard de notre contexte industriel, nous avons distingué différentes problématiques, divisées en trois chapitres de contribution. L'ensemble des propositions présentées dans ce manuscrit de thèse se sont articulées autour de l'outil de modélisation et simulation Papyrus.

Le chapitre 3 de ce manuscrit propose l'implémentation d'une extension Papyrus permettant l'externalisation de la gestion des risques industriels. Cette contribution est réalisée par la séparation des données d'entrée d'une simulation en un scénario de fonctionnement nominal, et un modèle de gestion des données relatives aux risques industriels. L'objectif de ce chapitre était de construire une base de données des risques, ayant un impact sur le déroulement de la simulation pendant la phase d''exécution. Cette base de données se matérialise par un ensemble de feuilles de calculs Excel, ou de tables de données. Au travers de notre outil de gestion des risques déporté, nous avons proposé la connexion d'un outil de simulation à une base de connaissances industrielles. Les connaissances engrangées par une entreprise (au moyen de tableurs Excel, bases de données, ou bigdata, réseaux de neurones, etc.) peuvent donc contribuer à l'anticipation de scénarios dans une démarche préventive (comme c'est le cas dans ce travail de recherche), ou dans une démarche spéculative, pour anticiper les évolutions d'un marché. Cette démarche s'inscrit dans la trajectoire d'une industrie 4.0 dirigée par les données.

Le chapitre 4 de ce manuscrit s'est inscrit dans une démarche de réutilisabilité et d'interopérabilité des systèmes. Il propose dans un premier temps la jonction de deux standards de simulation distribué (IEEE HLA 1516-2010 et FMI 2.0) afin de

permettre une communication entre des composants appartenant à chacun des deux standards. Dans un second temps, nous avons proposé l'ouverture de l'outil Papyrus au standard FMI afin de supporter la co-simulation de composants FMU. Ces composants pouvant désormais être chargés et exécutés par l'outil au cours de la simulation. Au travers de cette contribution, nous réduisons l'écart d'interopérabilité entre les outils des différents acteurs du contexte industriel. Dans notre cas, nous permettons la communication entre l'outil d'orchestration de simulation, et des composants externes (comme le FMU de relevés météorologiques temps réel). De manière plus générale, cette contribution est inscrite dans les enjeux d'interopérabilité de l'industrie 4.0. En permettant la collaboration et la communication entre des systèmes, nous permettons à l'industrie de gagner en agilité, de mieux maîtriser ses données, et d'augmenter ses capacités de traçabilité, ce qui rejoint la contribution précédente : une industrie dirigée par les données.

Les contributions du chapitre 5 de ce manuscrit ont eu pour objectif de piloter une simulation distribuée HLA. Cet objectif s'est divisé en deux ambitions. La première consistait en l'implémentation d'un mécanisme décideur/exécutant entre les différents fédérés qui composent une fédération HLA. Ainsi, un des composants distribués est désigné comme « décideur » et peut envoyer aux « exécutants » l'ordre de démarrer ou de stopper une simulation. La seconde ambition de ce chapitre consistait en l'ouverture de l'outil Papyrus au standard HLA afin de permettre la communication de l'outil avec les fédérés au travers du RTI. Ainsi, Papyrus peut maintenant être désigné comme étant le fédéré « décideur ». Il aura alors la capacité de diriger l'exécution des fédérés HLA connecté à la fédération.

Enfin, le chapitre 6 de ce manuscrit illustre l'assemblage des trois contributions. Chacune des fonctionnalités présentées dans les chapitres 3, 4 et 5 ont été l'objet d'une extension Papyrus. Ce dernier chapitre joint les extensions et cumule leurs effets au sein d'une application globale. Cette version étendue de Papyrus baptisée *DIAMANT*++ est un orchestrateur de simulation distribuée HLA – FMI prenant en compte la gestion des risques de façon externalisée. Au travers d'un exemple inspiré de notre contexte industriel, le comportement d'un système général est prédit en modélisant et en simulant les interactions du système avec des risques environnementaux (au travers d'un FMU), économiques, techniques (au travers d'un fichier Excel) et des sous-systèmes externes simulés de façon distribuée (au travers de JaamSim). Les différents processus nécessitant l'expertise de plusieurs domaines, l'entreprise est représentée par un scénario qui interconnecte des blocs de fonctionnalités ainsi que des interfaces assurant une connexion avec l'environnement extérieur.

Malgré les résultats prodigués dans cette thèse, des perspectives subsistent dans l'utilisation des standards de simulation distribuée HLA et FMI au sein de Papyrus. L'implémentation des mécanismes de gestion du temps coordonné entre HLA et Papyrus augmenterait les capacités d'échanges et de communications entre les deux outils. Actuellement, cinq outils distribués ont été développés pour l'application. Un objectif intéressant serait d'étendre le cadre de co-simulation construit pour intégrer d'autres composants orchestrés par le modèle Papyrus.

Nous avons démontré qu'il était possible de connecter un modèle Papyrus à des appels de services web. Transposer nos contributions à d'autres domaines (comme l'industrie des services) est une source de perspectives intéressantes. Une autre source d'évolution serait l'implémentation de mécanismes de simulation stochastique afin de représenter des systèmes industriels probabilistes. Enfin, une perspective d'évolution à long terme serait de connecter la base de données de gestion des risques à un système d'information d'entreprise pour dataminer et exploiter des gros volumes de données relatives à un contexte industriel.

1.Publications

L'ensemble des travaux de cette thèse a permis de rédiger les publications suivantes.

1.1.Revues internationales

- S. Gorecki, J. Ribault, G. Zacharewicz, Y. Ducq, and N. Perry, "Risk management and distributed simulation in Papyrus tool for decision making in industrial context," Comput. Ind. Eng., vol. 137, p. 106039, 2019, doi: https://doi.org/10.1016/j.cie.2019.106039.
- S. Gorecki, J. Possik, G. Zacharewicz, Y. Ducq, and N. Perry, "A Multi-Components Distributed Framework for Smart Production System Modeling & Simulation," Sustain. J. Spec. Issue Smart Prod. Oper. Manag. Ind. 40, 2020, 12, 6969. https://www.mdpi.com/2071-1050/12/17/6969

1.2.Conferences internationals

- S. Gorecki, G. Zacharewicz, and N. Perry, "Using High Level Architecture to combine simulations in company context mobile factory," I3M Barc., pp. 281–290, Sep. 2017.
- S. Gorecki, Y. Bouanan, G. Zacharewicz, and N. Perry, "BPMN Modeling for hla based simulation and visualization," in Society for Modeling and Simulation - SpringSim-Mod4Sim 2018, Baltimore, United States, Apr. 2018, pp. 1–12, Accessed: May 30, 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02136421.
- S. Gorecki, Y. Bouanan, G. Zacharewicz, J. Ribaud, and N. Perry, "Integrating HLA-Based Distributed Simulation for Management Science and BPMN," in INCOM 2018, Bergamo, Italy, Jun. 2018, pp. 1–5, Accessed: Jul. 15, 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02136377.
- Y. Bouanan, S. Gorecki, J. Ribault, G. Zacharewicz, and N. Perry, "Including in HLA Federation Functional Mockup Units for Supporting Interoperability and reusability in Distributed Simulation," Proceedings of the50th Computer Simulation Conference, Bordeaux, France, pp. 1–12, Jul. 2018.
- S. Gorecki, Y. Bouanan, J. Ribault, G. Zacharewicz, and N. Perry, "Including Co-Simulation in modeling and Simulation tool for supporting risk management in industrial context," International multidisciplinary modeling & simulation multiconference, Budapest, Hungary, Sep. 2018.

1.3.Conférences nationales

- S. Gorecki, G. Zacharewicz, and N. Perry, "Using High Level Architecture in the SEE Project for Industrial Context," Proc. SOHOMA 2017, vol. 762, pp. 281–290.

Chapitre 8
Bibliographie

[1] “Industrie 4.0 : définition et mise en œuvre vers l’usine connectée,” *Visiativ Solutions*. https://www.visiativ-solutions.fr/industrie-4-0/ (accessed Sep. 25, 2020).
[2] H. Kagermann, W. Wahlster, and J. Helbig, *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0–Deutschlands Zukunft als Industriestandort sichern, Forschungsunion Wirtschaft und Wissenschaft, Arbeitskreis Industrie 4.0*. Ort, 2013.
[3] A. Adamik and M. Nowicki, “Preparedness of companies for digital transformation and creating a competitive advantage in the age of Industry 4.0,” in *Proceedings of the International Conference on Business Excellence*, Poland, Lodz, 2018, vol. 12, no. 1, pp. 10–24.
[4] Å. Ericson, J. Lugnet, W. D. Solvang, H. Kaartinen, and J. Wenngren, “Challenges of Industry 4.0 in SME businesses,” in *2020 3rd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*, Jun. 2020, pp. 1–6, doi: 10.1109/SIMS49386.2020.9121542.
[5] A. Ghadge, M. Er Kara, H. Moradlou, and M. Goswami, “The impact of Industry 4.0 implementation on supply chains,” *Journal of Manufacturing Technology Management*, vol. 31, no. 4, pp. 669–686, Jan. 2020, doi: 10.1108/JMTM-10-2019-0368.
[6] A. M. Madni, C. C. Madni, and S. D. Lucero, “Leveraging digital twin technology in model-based systems engineering,” *Systems*, vol. 7, no. 1, p. 7, 2019.
[7] C. Mangles, “Gartner Hype Cycle 2018 – Most emerging technologies are 5-10 years away.” [Online]. Available: https://www.smartinsights.com/managing-digital-marketing/managing-marketing-technology/gartner-hype-cycle-2018-most-emerging-technologies-are-5-10-years-away/.
[8] S. J. E. Taylor, “Distributed simulation: state-of-the-art and potential for operational research,” *European Journal of Operational Research*, vol. 273, no. 1, pp. 1–19, Feb. 2019, doi: 10.1016/j.ejor.2018.04.032.
[9] Y. Sun, K. Bi, and S. Yin, “Measuring and Integrating Risk Management into Green Innovation Practices for Green Manufacturing under the Global Value Chain,” *Sustainability*, vol. 12, no. 2, p. 545, 2020.
[10] “COP 21 Paris France Sustainable Innovation Forum 2015 working with UNEP.” http://www.cop21paris.org/ (accessed Oct. 21, 2020).
[11] “Horizon 2020,” *Horizon 2020*. https://www.horizon2020.gouv.fr/ (accessed Oct. 21, 2020).
[12] “ALSOLEN HEAT SOLUTIONS FOR LIFE.” https://www.alsolen.com/fr/la-societe (accessed Oct. 27, 2020).
[13] F. Marmier, “Contribution au pilotage des projets et des processus par la prise en compte d’informations relatives aux activités, aux produits, aux ressources et aux risques,” PhD Thesis, 2014.
[14] S. Sperandio, “Usage de la modélisation multi-vue d’entreprise pour la conduite des systèmes de production,” PhD Thesis, 2005.
[15] F. Marle, “Modèles d’information et méthodes pour aider à la prise de décision en management de projets,” PhD Thesis, 2002.
[16] D. Genelot, *Manager dans la complexité: réflexions à l’usage des dirigeants*. INSEP editions, 2001.
[17] E. Tepeli, “Processus formalisé et systémique de management des risques par des projets de construction complexes et stratégiques,” PhD Thesis, Bordeaux, 2014.

[18] D. Gourc, “Vers un modèle général du risque pour le pilotage et la conduite des activités de biens et de services: Propositions pour une conduite des projets et une gestion des risques intégrées,” PhD Thesis, 2006.
[19] V. Giard and C. Midler, *Management et gestion de projet: bilan et perspectives*. GREGOR, IAE de Paris, 1996.
[20] “International Council on Systems Engineering Website.” https://www.incose.org/ (accessed Oct. 27, 2020).
[21] M. Fumey, “Méthode d’Evaluation des Risques Agrégés: application au choix des investissements de renouvellement d’installations,” PhD Thesis, 2001.
[22] Ministère de l’écologie, du développement et de l’aménagement durable, “Le plan de prévention des risques technologiques (PPRT).” [Online]. Available: https://www.ecologie.gouv.fr/sites/default/files/Guide_PPRT_tbd_complet.pdf.
[23] D. Breysse, *Maîtrise des risques en génie civil: Multiples dimensions des risques en génie civil*. Hermès Science publications, 2009.
[24] 14:00-17:00, “ISO 31000:2018,” *ISO*. https://www.iso.org/cms/render/live/fr/sites/isoorg/contents/data/standard/06/56/65694.html (accessed Sep. 15, 2020).
[25] 14:00-17:00, “IEC 31010:2019,” *ISO*. https://www.iso.org/cms/render/live/fr/sites/isoorg/contents/data/standard/07/21/72140.html (accessed Sep. 15, 2020).
[26] R. Atkinson, L. Crawford, and S. Ward, “Fundamental uncertainties in projects and the scope of project management,” *International journal of project management*, vol. 24, no. 8, pp. 687–698, 2006.
[27] T. H. Nguyen, “Contribution à la planification de projet: proposition d’un modèle d’évaluation des scénarios de risque-projet,” PhD Thesis, 2011.
[28] de l’Alimentation et de la M. Direction des Territoires, “Les 7 piliers de la prévention des risques,” Nov. 08, 2016. http://www.saint-pierre-et-miquelon.developpement-durable.gouv.fr/les-7-piliers-de-la-prevention-des-risques-r89.html (accessed Sep. 15, 2020).
[29] D. Kaslow, B. Ayres, P. T. Cahill, L. Hart, and R. Yntema, “A Model-Based Systems Engineering (MBSE) approach for defining the behaviors of CubeSats,” in *2017 IEEE Aerospace Conference*, 2017, pp. 1–14, doi: 10.1109/AERO.2017.7943865.
[30] G. S. Fishman, “Concepts and methods in discrete event digital simulation,” 1973.
[31] P. A. Fishwick and B. P. Zeigler, “A multimodel methodology for qualitative model engineering,” *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 2, no. 1, pp. 52–81, 1992.
[32] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018.
[33] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis*, vol. 3. McGraw-Hill New York, 2000.
[34] OMG Publication, *Unified Modeling Language (UML)*. 2017.
[35] A. A. A. Jilani, A. Nadeem, T. Kim, and E. Cho, “Formal Representations of the Data Flow Diagram: A Survey,” in *2008 Advanced Software Engineering and Its Applications*, 2008, pp. 153–158, doi: 10.1109/ASEA.2008.34.
[36] F. Ahmed, S. Robinson, and A. A. Tako, “Using the structred analysis and design technique (SADT) in simulation conceptual modeling,” in *Proceedings of the Winter Simulation Conference 2014*, 2014, pp. 1038–1049, doi: 10.1109/WSC.2014.7019963.
[37] H. Gruber, “Evaluation of Workflow Management Systems,” in *2009 IEEE Conference on Commerce and Enterprise Computing*, 2009, pp. 307–311, doi: 10.1109/CEC.2009.33.

[38] K. Ougaabal, G. Zacharewicz, Y. Ducq, and S. Tazi, “Visual Workflow Process Modeling and Simulation Approach Based on Non-Functional Properties of Resources,” *Applied Sciences*, vol. 10, no. 13, p. 4664, 2020.
[39] M. Geiger, S. Harrer, J. Lenhard, and G. Wirtz, “On the Evolution of BPMN 2.0 Support and Implementation,” in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2016, pp. 101–110, doi: 10.1109/SOSE.2016.39.
[40] J. Tyszer, *Object-oriented computer simulation of discrete-event systems*, vol. 10. Springer Science & Business Media, 2012.
[41] W. Kelton, R. Sadowski, and D. Sturrock, “Simulation with Arena (2007),” *Sydney: McGraw4Hill*, 2001.
[42] K. P. White and R. G. Ingalls, “Introduction to simulation,” in *2015 Winter Simulation Conference (WSC)*, 2015, pp. 1741–1755.
[43] B. P. Zeigler, “DEVS representation of dynamical systems: Event-based intelligent control,” *Proceedings of the IEEE*, vol. 77, no. 1, pp. 72–80, 1989.
[44] W. B. Nordgren, “Flexsim simulation environment,” in *Proceedings of the winter simulation conference*, 2002, vol. 1, pp. 250–252.
[45] A. Borshchev, S. Brailsford, L. Churilov, and B. Dangerfield, “Multi-method modelling: AnyLogic,” *Discrete-event simulation and system dynamics for management decision making*, pp. 248–279, 2014.
[46] A. Borshchev, Y. Karpov, and V. Kharitonov, “Distributed simulation of hybrid systems with AnyLogic and HLA,” *Future Generation Computer Systems*, vol. 18, no. 6, pp. 829–839, 2002.
[47] C. D. Pegden, “Introduction to SIMIO,” in *2008 Winter Simulation Conference*, 2008, pp. 229–235.
[48] D. W. Mc Gregor and M. J. Cain, “An introduction to SIMUL8,” *School of Mathematics & Statistics, University of Plymouth*, 2004.
[49] D. H. King and H. S. Harrison, “Open-source simulation software ‘JaamSim,’” in *2013 Winter Simulations Conference (WSC)*, 2013, pp. 2163–2171.
[50] J. Possik, “Contribution to a Methodology and a Co-Simulation Framework assessing the impact of Lean on Manufacturing Performance,” PhD Thesis, 2019.
[51] J. Göbel, P. Joschko, A. Koors, and B. Page, “The Discrete Event Simulation Framework DESMO-J: Review, Comparison To Other Frameworks And Latest Development.,” *ECMS*, vol. 13, pp. 100–109, 2013.
[52] F. Bergero and E. Kofman, “PowerDEVS: a tool for hybrid system modeling and real-time simulation,” *Simulation*, vol. 87, no. 1–2, pp. 113–132, 2011.
[53] T. Alix and G. Zacharewicz, “Product-service systems scenarios simulation based on G-DEVS/HLA: Generalized discrete event specification/high level architecture,” *Computers in Industry*, vol. 63, no. 4, pp. 370–378, 2012.
[54] “Eclipse Papyrus website.” [Online]. Available: https://www.eclipse.org/papyrus/.
[55] OMG Publication, *OMG System Modeling Language (SysML)*. 2019.
[56] OMG Publication, *Semantics of a Foundational Subset for Executable UML Models (fUML)*. 2018.
[57] S. Guermazi, J. Tatibouet, A. Cuccuru, S. Dhouib, and S. Gérard, “Executable Modeling with fUML and Alf in Papyrus: Tooling and Experiments,” *EXE MoDELS*, Ottawa, Canada, pp. 3–8, 2015.
[58] R. Ellner, S. Al-Hilank, H. Drexler, M. Jung, D. Kips, and M. Philippsen, “A FUML-Based Distributed Execution Machine for Enacting Software Process Models,” *ECMFA : Modeling Foundations and Application 2011*, p. pp 19-34, 2011.
[59] “About the Action Language for Foundational UML Specification Version 1.1.” https://www.omg.org/spec/ALF/About-ALF/ (accessed Oct. 26, 2020).

[60] S. Betz, S. Hickl, and A. Oberweis, “Risk management in global software development process planning,” in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2011, pp. 357–361.
[61] M. Perera and M. Ranasinghe, “Prompt list for risk management in Sri Lankan software industry,” in *2006 International Conference on Information and Automation*, 2006, pp. 7–12.
[62] R. D. Shachter, “Evaluating influence diagrams,” *Operations research*, vol. 34, no. 6, pp. 871–882, 1986.
[63] P. Sonchan and S. Ramingwong, “ARMI 2.0: An online risk management simulation,” in *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2015, pp. 1–5.
[64] E. Navarro and A. Van Der Hoek, “Multi-site evaluation of SimSE,” in *Proceedings of the 40th ACM technical symposium on Computer science education*, 2009, pp. 326–330.
[65] C. G. Von Wangenheim, R. Savi, and A. F. Borgatto, “SCRUMIA—An educational game for teaching SCRUM in computing courses,” *Journal of Systems and Software*, vol. 86, no. 10, pp. 2675–2687, 2013.
[66] J. Jürjens, *Secure systems development with UML*. Springer Science & Business Media, 2005.
[67] O. Altuhhova, R. Matulevicius, and A. Naved, “An Extension of BPMN for Security risk management,” *International Journal of Information System Modeling and Design*, p. pp 93-113, Jan. 2013.
[68] E. Dubois, P. Heumans, N. Mayer, and R. Matulevicius, “A Systematic Approach to Define the Domain of Information System Security Risk Management, edit dubois, Patrick Heymans,” *Intentional Perspectives on Information Systems Engineering*, pp. 289–306, 2010.
[69] N. Mayer, “Model-based management of information system security risk,” PhD Thesis, University of Namur, 2009.
[70] B. Marcinkowski and M. Kuciapski, “A Business Process Modeling Notation Extension for Risk Handling,” *International Conference on Computer Information Systems and Industrial Management*, pp. 374–381, Sep. 2012.
[71] M. Kuciapski, “Risk management in e-learning projects of courses development and implementation,” *Współczesna Gospodarka*, vol. 2, no. 1, pp. 63–75, 2011.
[72] M. Emes and B. Marcinkowski, “Risk-Oriented Modeling in Business Process Specification,” 2011.
[73] R. Laue and C. Müller, “The Business Process Simulation Standard (BPSIM): Chances And Limits.,” in *ECMS*, 2016, pp. 413–418.
[74] R. J. Scherer and S.-E. Schapke, “A distributed multi-model-based management information system for simulation and decision-making on construction projects,” *Advanced Engineering Informatics*, vol. 25, no. 4, pp. 582–599, 2011.
[75] K. Aksyonov, E. Bykov, and O. Aksyonova, “Development and application of software engineering solution BPsim. SD,” in *2013 European Modelling Symposium*, 2013, pp. 321–325.
[76] M. Fan, T. Thongsri, L. Axe, and T. A. Tyson, “Using a probabilistic approach in an ecological risk assessment simulation tool: test case for depleted uranium (DU),” *Chemosphere*, vol. 60, no. 1, pp. 111–125, 2005.
[77] H. Lu, L. Axe, and T. A. Tyson, “Development and application of computer simulation tools for ecological risk assessment,” *Environmental Modeling & Assessment*, vol. 8, no. 4, pp. 311–322, 2003.
[78] D. Bixio *et al.*, “A quantitative risk analysis tool for design/simulation of wastewater treatment plants,” *Water Science and Technology*, vol. 46, no. 4–5, pp. 301–307, 2002.

[79] F. Benaben *et al.*, “A conceptual framework and a suite of tools to support crisis management,” 2017.
[80] F. Marmier, I. F. Deniaud, and D. Gourc, “Strategic decision-making in NPD projects according to risk: Application to satellites design projects,” *Computers in Industry*, vol. 65, no. 8, pp. 1107–1114, 2014.
[81] A. Fertier, “Interprétation automatique de données hétérogènes pour la modélisation de situations collaboratives: application à la gestion de crise,” p. 183.
[82] Z. Tsiga, M. Emes, and A. Smith, “Implementation of a risk management simulation tool,” *Procedia computer science*, vol. 121, pp. 218–223, 2017.
[83] “The 4 Types of Projects in Project Management,” *Workfront*. https://www.workfront.com/blog/4-types-of-projects-which-kind-are-you-leading (accessed Sep. 15, 2020).
[84] E. Obeng, *All change!: The project leader’s secret handbook*. FT/Prentice Hall, 1994.
[85] M. Better, F. Glover, G. Kochenberger, and H. Wang, “Simulation optimization Applications in risk management,” *International Journal of Information Technology & Decision Making, vol. 7, No. 4*, pp. 571–587, 2008.
[86] “IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries,” *IEEE Std 610*, pp. 1–217, 1991, doi: 10.1109/IEEESTD.1991.106963.
[87] D. Chen, “Enterprise Interoperability Framework,” 2006.
[88] J. Ullberg, D. Chen, and P. Johnson, “Barriers to enterprise interoperability,” in *IFIP-International Workshop on Enterprise Interoperability*, 2009, pp. 13–24.
[89] A.-J. Berre *et al.*, “The ATHENA interoperability framework,” in *Enterprise interoperability II*, Springer, 2007, pp. 569–580.
[90] D. Chen, T. Knothe, and G. Doumeings, “POP* meta-model for enterprise model interoperability,” *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 175–180, 2009.
[91] V. Anaya *et al.*, “The Unified Enterprise Modelling Language – Overview and Further Work,” *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 11895–11906, Jan. 2008, doi: 10.3182/20080706-5-KR-1001.02014.
[92] F. B. Vernadat, “UEML: Towards a Unified Enterprise Modelling Language,” Troyes, France, France, 2001, Accessed: Sep. 15, 2020. [Online]. Available: https://hal.inria.fr/inria-00100684.
[93] IEEE Computer Society, *IEEE Standard 1516-2010 for M&S - HLA - Framework and Rules*. 2010.
[94] B. Vallespir, C. Braesch, V. Chapurlat, and D. Crestani, “L’intégration en modélisation d’entreprise: les chemins d’UEML,” 2003.
[95] B. Vallespir and G. Doumeingts, “La méthode GRAI,” *Groupement de Recherche en Productique (GRP), Ecole de printemps, la modélisation d’entreprise, Une réflexion sur l’enseignement et la pratique des méthodes, Albi*, pp. 28–30, 2002.
[96] “Model Driven Architecture (MDA) | Object Management Group.” https://www.omg.org/mda/ (accessed Sep. 15, 2020).
[97] J.-B. Chaudron, “Architecture de simulation distribuée temps-réel,” PhD Thesis, Toulouse, ISAE, 2012.
[98] R. M. Fujimoto, *Parallel and distributed simulation systems*, vol. 300. Wiley New York, 2000.
[99] “Centre technique et d’information de la défense Américaine.” [Online]. Available: https://discover.dtic.mil/.
[100] G. A. Wainer and K. Al-Zoubi, “An Introduction to Distributed Simulation,” in *Modeling and Simulation Fundamentals*, John Wiley & Sons, Ltd, 2010, pp. 373–402.

[101] F. Adelstein and M. Singhal, “Real-time causal message ordering in multimedia systems,” in *Telecommunication Systems - TELSYS*, 1997, vol. 7, pp. 36–43, doi: 10.1109/ICDCS.1995.500000.

[102] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 179–196.

[103] K. M. Chandy and J. Misra, “Distributed simulation: A case study in design and verification of distributed programs,” *IEEE Transactions on software engineering*, no. 5, pp. 440–452, 1979.

[104] R. E. Bryant, “Simulation of Packet Communication Architecture Computer Systems.,” MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1977.

[105] D. Jefferson, *Virtual Time, ACM Transactions on Programming Language and Systems*. 1985.

[106] A. Gafni, “Rollback mechanisms for optimistic distributed simulation systems,” *Distributed Simulation’88*, pp. 61–67, 1988.

[107] D. S. Committee, D. S. Committee, and others, “The DIS vision: A map to the future of distributed simulation,” *Institute for Simulation and Training*, 1994.

[108] D. C. Miller and J. A. Thorpe, “SIMNET: The advent of simulator networking,” *Proceedings of the IEEE*, vol. 83, no. 8, pp. 1114–1123, 1995.

[109] G. Zacharewicz, “Un environnement G-DEVS/HLA: Application à la modélisation et simulation distribuée de workflow,” PhD Thesis, 2006.

[110] W. Zhang, C.-L. Nie, Y.-L. Yu, J.-J. Liu, and K. Fu, “Study on conservative advancing-time algorithm in maintenance support simulation based on HLA,” *Xitong Fangzhen Xuebao / Journal of System Simulation*, vol. 24, pp. 2007–2011, 2012.

[111] X. Wang, S. J. Turner, M. Y. H. Low, and B. P. Gan, “Optimistic synchronization in HLA-based distributed simulation,” *Simulation*, vol. 81, no. 4, pp. 279–291, 2005.

[112] “The Portico Project.” http://porticoproject.org/ (accessed Sep. 15, 2020).

[113] “CERTI - Accueil [Savannah].” http://savannah.nongnu.org/projects/certi/ (accessed Oct. 10, 2020).

[114] “Home - MAK Technologies.” https://www.mak.com/ (accessed Sep. 15, 2020).

[115] “Pitch Technologies – Make Your Systems Work Together.” http://pitchtechnologies.com/ (accessed Sep. 15, 2020).

[116] A. Falcone, A. Garro, A. Anagnostou, N. R. Chaudhry, O.-A. Salah, and S. J. E. Taylor, “Easing the Development of HLA Federates The HLA Development Kit and its exploitation in the SEE Project,” *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Chengdu, China*, 2015.

[117] A. Falcone and A. Garro, “The SEE HLA Starter Kit: enabling the rapid prototyping of HLA-based simulations for space exploration,” in *Proceedings of the Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems 2016 (MSCIAAS 2016) and Space Simulation for Planetary Space Exploration (SPACE 2016)*, 2016, pp. 1–8.

[118] “Florida Space Institute - SEE Project,” *GitHub*. https://github.com/FlaSpaceInst (accessed Sep. 15, 2020).

[119] A. Falcone, A. Garro, A. D’Ambrogio, and A. Giglio, “Engineering Systems by combining BPMN and HLA-based distributed simulation,” *Systems Engineering Symposium, IEEE International, Vienna, Austria*, pp. 1–6, Oct. 2017.

[120] B. K. Choi, D. Lee, and D. H. Kang, “HLA/RTI-Based BPM Middleware for Collaborative Business Process Management,” in *International Conference on Business Process Management*, 2009, pp. 295–304.

[121] S. J. E. Taylor, J. S. Turner, and S. Strassburger, “Guidelines for commercial off-the-shelf simulation package interoperability,” *Winter Simulation Conference, Miami, USA*, pp. 193–204, Dec. 2008.

[122] P. Bocciarelli, D. Gianni, A. Pieroni, and A. D’Ambrogio, “A model-driven method for building distributed simulation systems from business process models,” *Simulation Conference (WSC), Proceedings of Winter, Berlin, Germany*, pp. 1–12, Dec. 2012.

[123] H. Bazoun, J. Ribault, G. Zacharewicz, Y. Ducq, and H. Boyé, “SLMTOOLBOX: Enterprise Service Process Modeling And Simulation By Coupling DEVS And Services Workflow,” *International Journal of Simulation and Process Modelling, Inderscience,* May 2016.

[124] G. Zacharewicz, C. Frydman, and N. Giambiasi, “G-DEVS/HLA environment for distributed simulations of workflows,” *Simulation*, vol. 84, no. 5, pp. 197–213, 2008.

[125] D. Lee, “Development of Mediator-Based Direct Wokrflow Simulation System and HLA/RTI-Based Collaborative BPMS Middleware,” PhD Thesis, PhD Thesis, Ph. D. Thesis, KAIST, South Korea, 2010.

[126] D. Chen, J. S. Turner, W. Cai, and M. Xiong, “A decoupled federate architecture for high level architecture-based distributed simulation,” *Journal of Parallel and Distributed Computing*, vol. 68, pp. 1487–1503, Nov. 2008.

[127] “Functional Mock-up Interface.” https://fmi-standard.org/ (accessed Sep. 15, 2020).

[128] “About | Functional Mock-up Interface.” https://fmi-standard.org/about/ (accessed Sep. 15, 2020).

[129] J. van der H. johan.van.der.heide[at]itea3.org, “MODELISAR,” *itea3.org*. https://itea3.org/project/modelisar.html (accessed Sep. 15, 2020).

[130] T. Blochwitz *et al.*, “Functional mockup interface 2.0: The standard for tool independent exchange of simulation models,” *Proceedings of the 9th International Modelica Conference, Munich, Germany*, pp. 173–184, Jan. 2012.

[131] J. Bastian, C. Clauß, S. Wolf, and P. Schneider, “Master for Co-Simulation Using FMI,” *Proceedings of the 8th International Modelica Conference*, pp. 115–120, Mar. 2011.

[132] N. Sievert, “Modelica Models in a Distributed Environment Using FMI and HLA.” 2016.

[133] F. Yilmaz, U. Durak, K. Taylan, and H. Oguztuzun, “Adapting Functional Mockup Units for HLA-compliant Distributed Simulation,” *10th International Modelica Conference*, Lund, Sweden, pp. 247–257, Mar. 2014.

[134] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, “Distributed Simulation of Heterogeneous and Real-Time Systems,” in *Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2013, pp. 55–62, doi: 10.1109/DS-RT.2013.14.

[135] R. Franke *et al.*, “Discrete-time models for control applications with FMI,” Jul. 2017, pp. 507–515, doi: 10.3384/ecp17132507.

[136] A. Garro and A. Falcone, “On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation,” in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, 2015, pp. 9–16.

[137] V. Savicks, M. Butler, and J. Colley, “Co-simulating Event-B and continuous models via FMI,” 2014.

[138] M. H. Spiegel, E. Widl, B. Heinzl, W. Kastner, and N. Akroud, “Model-Based Virtual Components in Event-Based Controls: Linking the FMI and IEC 61499,” *Applied Sciences*, vol. 10, no. 5, p. 1611, 2020.

[139] U. Pohlmann, W. Schäfer, H. Reddehase, J. Röckemann, and R. Wagner, “Generating functional mockup units from software specifications,” in

Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany, 2012, no. 076, pp. 765–774.

[140] P. Saxena, P. Stavropoulos, J. Kechagias, and K. Salonitis, “Sustainability Assessment for Manufacturing Operations,” *Energies*, vol. 13, no. 11, 2020, doi: 10.3390/en13112730.

[141] J. J. Possik, A. A. Amrani, and G. Zacharewicz, “Development of a co-simulation system as a decision-aid in Lean tools implementation,” in *Proceedings of the 50th Computer Simulation Conference*, 2018, p. 21.

Chapitre 9
Annexes

Code Advice Papyrus Gestion des risques

```
package org.eclipse.papyrus.diamant.engine;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import org.eclipse.papyrus.diamant.FailureProfile.DbActionFailure;
import org.eclipse.papyrus.moka.constrainedexecution.metamodel.constrainedexecution.util.DurationHelper;
import org.eclipse.papyrus.moka.externalcontrol.advice.BasicControllerAdvice;
import org.eclipse.papyrus.moka.fuml.Semantics.Actions.BasicActions.IActionActivation;
import org.eclipse.uml2.uml.Action;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class AdviceDbActionFailure extends BasicControllerAdvice<Action, IActionActivation> {

	private FailureManager manager;
	private DbActionFailure failableAction;
	private boolean canstart = true;

	public AdviceDbActionFailure(Action element, IActionActivation delegatedVisitor, DbActionFailure failableAction,
FailureManager manager) {
		super(element, delegatedVisitor);
		this.failableAction =failableAction;
		this.manager = manager;
		manager.resetFailableElement(failableAction);
	}

	@Override
	public boolean canStart() {
		canstart = manager.canStart(failableAction);
		return true;
	}

	@Override
	public boolean canFinish() {
		return super.canFinish();
	}

	@Override
	public void doStartAction() {
		 manager.incrementFailableElement(failableAction);
	}

	@Override
	public void doFinishAction() {
		super.doFinishAction();
	}
```

```
	@Override
	public Double getDuration() {
		if(!canstart) {
			String name = failableAction.getBase_Action().getName();
			String durationString = "1m";

			try
			{
				XlsxReader myBD = new XlsxReader(this.manager.filePath,
this.manager.sheetName);

				//get actual date in simulation
				String dateEvent = getCurrentDate("dd/MM/yyyy HH:mm:ss");

				//Write simulation time in Excel file
				myBD.setFailureDate(dateEvent, failableAction.getID());

				//Get task imact
				durationString = myBD.getCellAttribute(failableAction.getID());

				manager.logger.log(failableAction.getID() + " : " + durationString + " at " +
dateEvent);
			}catch (Exception e)
			{
				manager.logger.log("AdviceDbActionFailure : " + e.toString() + " : " +
e.getMessage());
				e.printStackTrace();
			}

			//we check we have a correct duration (same syntax as Task duration)
			if (durationString == null || !DurationHelper.isValidString(durationString))
				durationString = "1m";

			//Convert duration in ns
			long duration = DurationHelper.getDurationNanos(durationString);
			return (double) duration;
		}
		return super.getDuration();//(double) DurationHelper.getDurationNanos("96h");
	}

	private String getCurrentDate(String format)
	{
		double currentTime = manager.getCurrentTime();

		currentTime /= 1000000;
		Date date = new Date((long)currentTime);
		SimpleDateFormat sdf = new SimpleDateFormat(format);

		return sdf.format(date);
	}
}

package org.eclipse.papyrus.diamant.engine;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.SimpleFormatter;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileSystemView;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

import org.eclipse.papyrus.moka.constrainedexecution.engine.engine.ConstrainedBPMNExecutionEngine;
import org.eclipse.papyrus.moka.discreteevent.DEScheduler;
import org.eclipse.papyrus.moka.externalcontrol.advice.IControllerAdviceFactory;

public class EngineExample extends ConstrainedBPMNExecutionEngine {
```

```
	private FailureManager manager ;
	public Logger logger;

	@Override
	protected List<IControllerAdviceFactory> getAdviceFactories() {
		List<IControllerAdviceFactory> factories = new ArrayList<IControllerAdviceFactory>();
		factories.addAll(super.getAdviceFactories());

		//we return our new custom manager
		factories.add(new FailureAdviceFactory(manager));

		return factories;
	}

	@Override
	protected void doPreRunActions() {
		//init logger
		logger = new Logger();

		String targetDir = FileSystemView.getFileSystemView().getHomeDirectory().toString();
		String mode_auto = "disable";
		String targetFile = "";
		String sheetName = "Diamant";
		String filePath;

		if(!new File("config.json").exists())
		{
			logger.log("config.JSON not found");

			JSONObject obj = new JSONObject();
			obj.put("mode_auto", "disable");
			obj.put("targetFile", "C:/ownCloud/ALSOLEN_TECH/Diamant.xlsx");
			obj.put("targetDir", "C:/ownCloud/ALSOLEN_TECH");
			obj.put("sheetName", "Diamant");
			try
			{
				FileWriter file = new FileWriter("config.json");
				file.write(obj.toJSONString());
				file.flush();
				logger.log("config.JSON created");
			} catch(Exception e) {logger.log("EngineExample : " + e.toString() + " : " + e.getMessage());
e.printStackTrace();}
		}

		JSONParser parser = new JSONParser();
		try
		{
			logger.log("Reading config.JSON...");
			Object obj = parser.parse(new FileReader("config.json"));
			JSONObject jsonObject = (JSONObject) obj;

			targetDir = (String) jsonObject.get("targetDir");
			mode_auto = (String) jsonObject.get("mode_auto");
			targetFile = (String) jsonObject.get("targetFile");
			sheetName = (String) jsonObject.get("sheetName");

		} catch (Exception e) {logger.log("EngineExample : " + e.toString() + " : " +
e.getMessage());e.printStackTrace();}

		if(mode_auto.contains("enable"))
		{
			filePath = targetFile;
		} else
		{
			//popup fenetre selection fichier excel
			JFileChooser jfc = new JFileChooser(targetDir);
			filePath = targetFile;

			int returnValue = jfc.showOpenDialog(null);
			if (returnValue == JFileChooser.APPROVE_OPTION)
				filePath = jfc.getSelectedFile().getAbsolutePath();
			else
				logger.log("erreur chargement fichier excel");
		}
```

```
                super.doPreRunActions();
                manager = new FailureManager(DEScheduler.getInstance(), filePath, sheetName, logger);
        }


        @Override
        protected void doPostRunActions() {
                logger.log("simulation finished");
                logger.print();

                super.doPostRunActions();
        }
}
```

Code Advice Papyrus gestion FMI

```
package org.eclipse.papyrus.diamant.engine;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import org.eclipse.papyrus.diamant.FMIProfile.FMIConnexion;
import org.eclipse.papyrus.moka.constrainedexecution.metamodel.constrainedexecution.util.DurationHelper;
import org.eclipse.papyrus.moka.externalcontrol.advice.BasicControllerAdvice;
import org.eclipse.papyrus.moka.fuml.Semantics.Actions.BasicActions.IActionActivation;
import org.eclipse.uml2.uml.Action;
import org.javafmi.wrapper.Simulation;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class AdviceDbActionFMULoader extends BasicControllerAdvice<Action, IActionActivation> {

	private FMUManager manager;
	private FMIConnexion failableAction;
	private boolean canstart = true;
	private String pathfile = "RiskManagement.fmu";

	public AdviceDbActionFMULoader(Action element, IActionActivation delegatedVisitor, FMIConnexion failableAction,
FMUManager manager) {
		super(element, delegatedVisitor);
		this.failableAction =failableAction;
		this.manager = manager;
	}

	@Override
	public boolean canStart() {
		return true;
	}

	@Override
	public boolean canFinish() {
		return super.canFinish();
	}

	@Override
	public void doFinishAction() {
		super.doFinishAction();
	}

	@Override
	public Double getDuration() {
		if(!canstart) {
			String name = failableAction.getBase_Action().getName();
			double simuluationImact = 0.0;
			try
			{
				//simulation FMU step forward

	this.manager.simulation.write("RM.SimTime").with(this.manager.getCurrentTime());
				this.manager.simulation.doStep(this.manager.jsonReader.getFMUTimeStep());
				simuluationImact = this.manager.simulation.read("RM.impact").asDouble();

				manager.logger.log(failableAction.getID() + " : " + simuluationImact.toString() + "
at " + dateEvent);
			}catch (Exception e)
			{
				manager.logger.log("AdviceDbActionFMULoader : " + e.toString() + " : " +
e.getMessage());
				e.printStackTrace();
			}

			return simuluationImact;
		}
		return super.getDuration();//(double) DurationHelper.getDurationNanos("96h");
	}
```

```
	private String getCurrentDate(String format)
	{
		double currentTime = manager.getCurrentTime();

		currentTime /= 1000000;
		Date date = new Date((long)currentTime);
		SimpleDateFormat sdf = new SimpleDateFormat(format);

		return sdf.format(date);
	}
}
```

```
package org.eclipse.papyrus.diamant.engine;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.SimpleFormatter;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileSystemView;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

import org.eclipse.papyrus.moka.constrainedexecution.engine.engine.ConstrainedBPMNExecutionEngine;
import org.eclipse.papyrus.moka.discreteevent.DEScheduler;
import org.eclipse.papyrus.moka.externalcontrol.advice.IControllerAdviceFactory;

public class EngineExample extends ConstrainedBPMNExecutionEngine {

	private FMUManager manager ;
	public Logger logger;

	@Override
	protected List<IControllerAdviceFactory> getAdviceFactories() {
		List<IControllerAdviceFactory> factories = new ArrayList<IControllerAdviceFactory>();
		factories.addAll(super.getAdviceFactories());

		//we return our new custom manager
		factories.add(new FMUAdviceFactory(manager));

		return factories;
	}

	@Override
	protected void doPreRunActions() {
		//init logger
		logger = new Logger();

		super.doPreRunActions();
		manager = new FMUManager(DEScheduler.getInstance(), filePath, sheetName, logger);
	}

	@Override
	protected void doPostRunActions() {
		//closing FMU simulation
		manager.simulation.terminate();
		logger.log("simulation finished");
		logger.print();

		super.doPostRunActions();
	}
}
```

```
package org.eclipse.papyrus.diamant.engine;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;
import java.util.logging.Level;

import org.eclipse.emf.ecore.EObject;
import org.eclipse.papyrus.diamant.FMIProfile.ActionLauncherFMU;
import org.eclipse.papyrus.moka.constrainedexecution.metamodel.constrainedexecution.util.DurationHelper;
import org.eclipse.papyrus.moka.constrainedexecution.profile.constrainedexecutionprofile.ExecutableElementStereo;
import org.eclipse.papyrus.moka.discreteevent.DEScheduler;
import org.eclipse.papyrus.moka.discreteevent.Event;
import org.eclipse.uml2.uml.util.UMLUtil;

public class FMUManager {
	DEScheduler scheduler;
	public Logger logger;
	public JSONReader jsonReader;
	public Simulation simulation;

	FMUManager(DEScheduler scheduler, String filePath, String sheetName, Logger logger) {
		this.scheduler = scheduler;
		this.filePath = filePath;
		this.sheetName = sheetName;
		this.logger = logger;
		this.jsonReader = new JSONReader("configSimulation.json");

		//init FMU simulation
		simulation = new Simulation(jsonReader.getFMUFilePath());
		simulation.init(0,30);
		simulation.write("RM.key").with(jsonReader.getAPIKey()); //clé API OpenWeatherMap
		simulation.write("RM.location").with(location);
	}

	public double getCurrentTime()
	{
		return this.scheduler.getCurrentTime();
	}
}
```

Code Advice Papyrus Gestion HLA

```
package org.eclipse.papyrus.diamant.engine;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import org.eclipse.papyrus.diamant.FMIProfile.FederateConnexion;
import org.eclipse.papyrus.moka.constrainedexecution.metamodel.constrainedexecution.util.DurationHelper;
import org.eclipse.papyrus.moka.externalcontrol.advice.BasicControllerAdvice;
import org.eclipse.papyrus.moka.fuml.Semantics.Actions.BasicActions.IActionActivation;
import org.eclipse.uml2.uml.Action;
import org.javafmi.wrapper.Simulation;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class AdviceDbActionFederateLoader extends BasicControllerAdvice<Action, IActionActivation> {

	private HLAManager manager;
	private FederateConnexion failableAction;
	private boolean canstart = true;

	public AdviceDbActionFederateLoader(Action element, IActionActivation delegatedVisitor, FederateConnexion failableAction, HLAManager manager) {
		super(element, delegatedVisitor);
		this.failableAction =failableAction;
		this.manager = manager;
	}

	@Override
	public boolean canStart() {
		return true;
	}

	@Override
	public boolean canFinish() {
		return super.canFinish();
	}

	@Override
	public void doFinishAction() {
		super.doFinishAction();
	}

	@Override
	public Double getDuration() {
		if(!canstart) {
			String name = failableAction.getBase_Action().getName();
			double simuluationImact = 0.0;
			try
			{
				//Exel risque management
				XlsxReader myBD = new XlsxReader(this.manager.filePath, this.manager.sheetName);

				//get actual date in simulation
				String dateEvent = getCurrentDate("dd/MM/yyyy HH:mm:ss");

				//Write simulation time in Excel file
				myBD.setFederateFailureDate(dateEvent, failableAction.getFederateName());

				//Get task imact
				Map <String, String> imactsOnAttributes = myBD.getFederateAttribute(failableAction.getFederateName());

				for (Entry<String, String> e : imactsOnAttributes.entrySet()) {
				 String parameter = imactsOnAttributes.getKey();
				 String value = imactsOnAttributes.getValue();

				 //modifing task impact on federate scenario
				 manager.logger.log("[parameter] : " + value);

if(!manager.jsonReader.setSimulationParameter(failableAction.getFederateName(), parameter, value))
```

```
                                                manager.logger.log("Erreur de saisie via "
+failableAction.getFederateName()+"["+parameter+"]");
                                        }


                                        //Federate execution
                                        this.manager.jsonReader.writeSimTime(failableAction.getFederateName(),

                                this.manager.getCurrentTime());
                                        double simuluationImact =
this.manager.masterFederate.start(failableAction.getFederateName())

                                        manager.logger.log(failableAction.getID() + " : " + simuluationImact.toString() + "
at " + dateEvent);
                                }catch (Exception e)
                                {
                                        manager.logger.log("AdviceDbActionFederateLoader : " + e.toString() + " : " +
e.getMessage());
                                        e.printStackTrace();
                                }

                                return simuluationImact;
                        }
                        return super.getDuration();//(double) DurationHelper.getDurationNanos("96h");
                }

                private String getCurrentDate(String format)
                {
                        double currentTime = manager.getCurrentTime();

                        currentTime /= 1000000;
                        Date date = new Date((long)currentTime);
                        SimpleDateFormat sdf = new SimpleDateFormat(format);

                        return sdf.format(date);
                }
}
```

Fédéré décideur Papyrus

```
import hla.rti1516e.*;
import hla.rti1516e.encoding.HLAunicodeString;
import hla.rti1516e.NullFederateAmbassador;
import hla.rti1516e.encoding.DecoderException;
import hla.rti1516e.encoding.EncoderFactory;
import hla.rti1516e.encoding.HLAfloat32LE;
import hla.rti1516e.encoding.HLAinteger32LE;
import hla.rti1516e.exceptions.*;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;

class MasterFederate extends NullFederateAmbassador {
 private RTIambassador _rtiAmbassador;
 private final String ip;
 private final String federateName;
 //InteractionClassHandle
		//Message
 private InteractionClassHandle InteractionMessage;
 private ParameterHandle _parameterId;
 private ParameterHandle _parameterDst;
 private ParameterHandle _parameterMsg;
 private ParameterHandle _parameterConfirm;

		//FederateEnd
 private InteractionClassHandle InteractionFederateEnd;
 private ParameterHandle _parameterFederateName;
 private ParameterHandle _parameterFederateDuration;

 //ObjectInstanceHandle
		//KPI
 private ObjectInstanceHandle ObjectKPIHandler;
 private AttributeHandle _attributeKPIFederateName;
 private AttributeHandle _attributeKPISimTime;
 private AttributeHandle _attributeLEADTIME;
 private AttributeHandle _attributeWIP;

		//Scenario_Appro
 private ObjectInstanceHandle ObjectScenario_ApproHandler;
 private AttributeHandle _attributeScenario_ApproFederate_Name;
 private AttributeHandle _attributeScenario_ApproSimTime;
 private AttributeHandle _attributeN_HPriority;
 private AttributeHandle _attributeN_LPriority;
 private AttributeHandle _attributeF_HPriority;
 private AttributeHandle _attributeF_LPriority;
 private AttributeHandle _attributeD_Loading;
 private AttributeHandle _attributeD_TR;
 private AttributeHandle _attributeD_VOL;
 private AttributeHandle _attributeD_TR2;
 private AttributeHandle _attributeD_Depal;

		//Scenario_Deploy
 private ObjectInstanceHandle ObjectScenario_DeployHandler;
 private AttributeHandle _attributeScenario_DeployFederateName;
 private AttributeHandle _attributeScenario_DeploySimTime;
 private AttributeHandle _attributeD_Soudure1;
 private AttributeHandle _attributeD_Soudure2;
 private AttributeHandle _attributeD_Portique1;
 private AttributeHandle _attributeD_Portique2;
 private AttributeHandle _attributeD_Colle1;
 private AttributeHandle _attributeD_Colle2;
 private AttributeHandle _attributeD_Mirror1;
 private AttributeHandle _attributeD_Mirror2;
 private AttributeHandle _attributeD_Assemblage;
 private AttributeHandle _attributeD_test;
 private AttributeHandle _attributeD_Pose;
 private AttributeHandle _attributeD_install;

 private manager.jsonReader manager.jsonReader;
```

```
private String _username;
private int nbWaitingFederates = 3;
private double LoopScenario_Appro = Double.NaN;
private double LoopScenario_Deploy = Double.NaN;
public boolean shutdown = true;

private volatile boolean _reservationComplete;
private volatile boolean _reservationSucceeded;
private final Object _reservationSemaphore = new Object();

private static final int CRC_PORT = 8989;
private static final String FEDERATION_NAME = "PapyrusFederation";
private EncoderFactory _encoderFactory;
private String configFilePath;

public MasterFederate(String ip, String federateName, String configFilePath, FederateManager manager)
{
 this.ip = ip;
 this.federateName = federateName;
 this.configFilePath = configFilePath;
 run();
}

void run()
{
 try {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

  try {
                //get rtiFactory
                RtiFactory rtiFactory = RtiFactoryFactory.getRtiFactory();
                _rtiAmbassador = rtiFactory.getRtiAmbassador();
                _encoderFactory = rtiFactory.getEncoderFactory();
  } catch (Exception e) {
                System.out.println("Unable to create RTI ambassador.");
                e.printStackTrace();
                return;
  }

  String settingsDesignator = "crcAddress=" + ip;
  //connect to RTI
  _rtiAmbassador.connect(this, CallbackModel.HLA_IMMEDIATE, settingsDesignator);

  try {
  // Clean up old federation
  _rtiAmbassador.destroyFederationExecution(FEDERATION_NAME);
  } catch (FederatesCurrentlyJoined ignored) {
  } catch (FederationExecutionDoesNotExist ignored) {
  }
  File fddFile = new File("Communication.xml");
  try {
                //create federation
  _rtiAmbassador.createFederationExecution(FEDERATION_NAME, new URL[] { fddFile.toURL() }, "HLAfloat64Time");
  } catch (FederationExecutionAlreadyExists ignored) {
  }

  //connect to federation
  _rtiAmbassador.joinFederationExecution(federateName, FEDERATION_NAME, new URL[]{fddFile.toURL()});

  // Subscribe and publish interactions
              //Message
  InteractionMessage = _rtiAmbassador.getInteractionClassHandle("Message");
  _parameterId = _rtiAmbassador.getParameterHandle(InteractionMessage, "Id");
  _parameterDst = _rtiAmbassador.getParameterHandle(InteractionMessage, "Dst");
  _parameterMsg = _rtiAmbassador.getParameterHandle(InteractionMessage, "Msg");
  _parameterConfirm = _rtiAmbassador.getParameterHandle(InteractionMessage, "Confirm");

  _rtiAmbassador.subscribeInteractionClass(InteractionMessage);
  _rtiAmbassador.publishInteractionClass(InteractionMessage);

                //FederateEnd
  InteractionFederateEnd = _rtiAmbassador.getInteractionClassHandle("FederateEnd");
  _parameterFederateName = _rtiAmbassador.getParameterHandle(InteractionFederateEnd, "FederateName");
  _parameterFederateDuration = _rtiAmbassador.getParameterHandle(InteractionFederateEnd, "FederateDuration");
```

```
_rtiAmbassador.subscribeInteractionClass(InteractionFederateEnd);
_rtiAmbassador.publishInteractionClass(InteractionFederateEnd);

// Subscribe and publish objects
        //KPI
ObjectClassHandle ObjectKPI = _rtiAmbassador.getObjectClassHandle("KPI");
AttributeHandleSet KPIAttributeSet = _rtiAmbassador.getAttributeHandleSetFactory().create();
_attributeKPIFederateName = _rtiAmbassador.getAttributeHandle(ObjectKPI, "FederateName");
_attributeKPISimTime = _rtiAmbassador.getAttributeHandle(ObjectKPI, "SimTime");
_attributeLEADTIME = _rtiAmbassador.getAttributeHandle(ObjectKPI, "LEADTIME");
_attributeWIP = _rtiAmbassador.getAttributeHandle(ObjectKPI, "WIP");

KPIAttributeSet.add(_attributeKPIFederateName);
KPIAttributeSet.add(_attributeKPISimTime);
KPIAttributeSet.add(_attributeLEADTIME);
KPIAttributeSet.add(_attributeWIP);

_rtiAmbassador.subscribeObjectClassAttributes(ObjectKPI, KPIAttributeSet);
_rtiAmbassador.publishObjectClassAttributes(ObjectKPI, KPIAttributeSet);

        //Scenario_Appro
ObjectClassHandle ObjectScenario_Appro = _rtiAmbassador.getObjectClassHandle("Scenario_Appro");
AttributeHandleSet Scenario_ApproAttributeSet = _rtiAmbassador.getAttributeHandleSetFactory().create();
_attributeScenario_ApproFederate_Name = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "Federate_name");
_attributeScenario_ApproSimTime = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "SimTime");
_attributeN_HPriority = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "N_HPriority");
_attributeN_LPriority = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "N_LPriority");
_attributeF_HPriority = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "F_HPriority");
_attributeF_LPriority = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "F_LPriority");
_attributeD_Loading = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "D_Loading");
_attributeD_TR = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "D_TR");
_attributeD_VOL = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "D_VOL");
_attributeD_TR2 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "D_TR2");
_attributeD_Depal = _rtiAmbassador.getAttributeHandle(ObjectScenario_Appro, "D_Depal");

Scenario_ApproAttributeSet.add(_attributeScenario_ApproFederate_Name);
Scenario_ApproAttributeSet.add(_attributeScenario_ApproSimTime);
Scenario_ApproAttributeSet.add(_attributeN_HPriority);
Scenario_ApproAttributeSet.add(_attributeN_LPriority);
Scenario_ApproAttributeSet.add(_attributeF_HPriority);
Scenario_ApproAttributeSet.add(_attributeF_LPriority);
Scenario_ApproAttributeSet.add(_attributeD_Loading);
Scenario_ApproAttributeSet.add(_attributeD_TR);
Scenario_ApproAttributeSet.add(_attributeD_VOL);
Scenario_ApproAttributeSet.add(_attributeD_TR2);
Scenario_ApproAttributeSet.add(_attributeD_Depal);

_rtiAmbassador.subscribeObjectClassAttributes(ObjectScenario_Appro, Scenario_ApproAttributeSet);
_rtiAmbassador.publishObjectClassAttributes(ObjectScenario_Appro, Scenario_ApproAttributeSet);

        //Scenario_Deploy
ObjectClassHandle ObjectScenario_Deploy = _rtiAmbassador.getObjectClassHandle("Scenario_Deploy");
AttributeHandleSet Scenario_DeployAttributeSet = _rtiAmbassador.getAttributeHandleSetFactory().create();
_attributeScenario_DeployFederateName = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "Federate_name");
_attributeScenario_DeploySimTime = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "SimTime");
_attributeD_Soudure1 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Soudure1");
_attributeD_Soudure2 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Soudure2");
_attributeD_Portique1 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Portique1");
_attributeD_Portique2 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Portique2");
_attributeD_Colle1 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Colle1");
_attributeD_Colle2 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Colle2");
_attributeD_Mirror1 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Mirror1");
_attributeD_Mirror2 = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Mirror2");
_attributeD_Assemblage = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Assemblage");
_attributeD_test = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_test");
_attributeD_Pose = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_Pose");
_attributeD_install = _rtiAmbassador.getAttributeHandle(ObjectScenario_Deploy, "D_install");

Scenario_DeployAttributeSet.add(_attributeScenario_DeployFederateName);
Scenario_DeployAttributeSet.add(_attributeScenario_DeploySimTime);
Scenario_DeployAttributeSet.add(_attributeD_Soudure1);
Scenario_DeployAttributeSet.add(_attributeD_Soudure2);
Scenario_DeployAttributeSet.add(_attributeD_Portique1);
Scenario_DeployAttributeSet.add(_attributeD_Portique2);
Scenario_DeployAttributeSet.add(_attributeD_Colle1);
```

```
Scenario_DeployAttributeSet.add(_attributeD_Colle2);
Scenario_DeployAttributeSet.add(_attributeD_Mirror1);
Scenario_DeployAttributeSet.add(_attributeD_Mirror2);
Scenario_DeployAttributeSet.add(_attributeD_Assemblage);
Scenario_DeployAttributeSet.add(_attributeD_test);
Scenario_DeployAttributeSet.add(_attributeD_Pose);
Scenario_DeployAttributeSet.add(_attributeD_install);

_rtiAmbassador.subscribeObjectClassAttributes(ObjectScenario_Deploy, Scenario_DeployAttributeSet);
_rtiAmbassador.publishObjectClassAttributes(ObjectScenario_Deploy, Scenario_DeployAttributeSet);

// Reserve object instance name and register object instance
do {
_username = federateName;

try {
 _reservationComplete = false;
 _rtiAmbassador.reserveObjectInstanceName(_username);
 synchronized (_reservationSemaphore) {
  // Wait for response from RTI
  while (!_reservationComplete) {
   try {
   _reservationSemaphore.wait();
   } catch (InterruptedException ignored) {
   }
  }
 }
 if (!_reservationSucceeded) {
  System.out.println("Name already taken, try again.");
 }
} catch (IllegalName e) {
 System.out.println("Illegal name. Try again.");
} catch (RTIexception e) {
 System.out.println("RTI exception when reserving name: " + e.getMessage());
 return;
}
} while (!_reservationSucceeded);

ObjectKPIHandler = _rtiAmbassador.registerObjectInstance(ObjectKPI, _username);
ObjectScenario_ApproHandler = _rtiAmbassador.registerObjectInstance(ObjectScenario_Appro, _username);
ObjectScenario_DeployHandler = _rtiAmbassador.registerObjectInstance(ObjectScenario_Deploy, _username);

System.out.println("Papyrus federate online. Waiting for federates");
manager.jsonReader = new manager.jsonReader(configFilePath);


while(nbWaitingFederates != 0) {
		Thread.sleep(1000);
}

System.out.println("All federates are connected, waiting for launching.");

while(shutdown)
{
		Thread.sleep(4000);
}

System.out.println("Shutting down federation");

_rtiAmbassador.resignFederationExecution(ResignAction.DELETE_OBJECTS_THEN_DIVEST);
try {
_rtiAmbassador.destroyFederationExecution(FEDERATION_NAME);
} catch (FederatesCurrentlyJoined ignored) {
}
_rtiAmbassador.disconnect();
_rtiAmbassador = null;
} catch (Exception e) {
e.printStackTrace();
			try{
				System.out.println("Press <ENTER> to shutdown");
				BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
				in.readLine();
			} catch (IOException ioe){
			}
}
```

```
}

@Override
public void receiveInteraction(InteractionClassHandle interactionClass,
      ParameterHandleValueMap theParameters,
      byte[] userSuppliedTag,
      OrderType sentOrdering,
      TransportationTypeHandle theTransport,
      SupplementalReceiveInfo receiveInfo)
 throws
 FederateInternalError
{
 if (interactionClass.equals(InteractionMessage)) {
  try {
  HLAunicodeString IdDecoder = _encoderFactory.createHLAunicodeString();
  HLAunicodeString DstDecoder = _encoderFactory.createHLAunicodeString();
  HLAunicodeString MsgDecoder = _encoderFactory.createHLAunicodeString();
  HLAunicodeString ConfirmDecoder = _encoderFactory.createHLAunicodeString();
  IdDecoder.decode(theParameters.get(_parameterId));
  DstDecoder.decode(theParameters.get(_parameterDst));
  MsgDecoder.decode(theParameters.get(_parameterMsg));
  ConfirmDecoder.decode(theParameters.get(_parameterConfirm));
  String Id = IdDecoder.getValue();
  String Dst = DstDecoder.getValue();
  String Msg = MsgDecoder.getValue();
  String Confirm = ConfirmDecoder.getValue();

  //check if federate is waited
  if(manager.jsonReader.checkIfFederateIsWaited(Id) && Dst.contains("Master") && Msg.contains("join")){
            System.out.println(Id + " has joined the federation. Wait for launch");
            nbWaitingFederates--;
  }
  } catch (DecoderException e) {
  System.out.println("Failed to decode incoming interaction");
  }
 }
 if (interactionClass.equals(InteractionFederateEnd))
 {
            try{
                        HLAunicodeString FederateNameDecoder = _encoderFactory.createHLAunicodeString();
                        HLAfloat32LE FederateDurationDecoder = _encoderFactory.createHLAfloat32LE();
                        FederateNameDecoder.decode(theParameters.get(_parameterFederateName));
                        FederateDurationDecoder.decode(theParameters.get(_parameterFederateDuration));
                        String FederateName = FederateNameDecoder.getValue();
                        double FederateDuration = FederateDurationDecoder.getValue();

                        if(FederateName.equals("Scenario_Appro"))
                                    LoopScenario_Appro = FederateDuration;
                        else if (FederateName.equals("Scenario_Deploy"))
                                    LoopScenario_Deploy = FederateDuration;
            } catch (DecoderException e){
                        e.printStackTrace();
            }
 }
}

public double start(String federateName)
{
            try{
                        //loading simulation parameters
                        if(federateName.equals("Scenario_Appro"))
                        {
                                    AttributeHandleValueMap attributeValues =
_rtiAmbassador.getAttributeHandleValueMapFactory().create(1);

                                    HLAunicodeString Federate_nameEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "Federate_name"));
                                    attributeValues.put(_attributeScenario_ApproFederate_Name,
Federate_nameEncoder.toByteArray());
                                    HLAfloat32LE SimTimeEncoder =
_encoderFactory.createHLAfloat32LE(Float.parseFloat(manager.jsonReader.getSimulationParameter(federateName,
"SimTime")));
                                    attributeValues.put(_attributeScenario_ApproSimTime, SimTimeEncoder.toByteArray());
```

```
                HLAfloat32LE N_HPriorityEncoder =
_encoderFactory.createHLAfloat32LE(Float.parseFloat(manager.jsonReader.getSimulationParameter(federateName,
"N_HPriority")));
                attributeValues.put(_attributeN_HPriority, N_HPriorityEncoder.toByteArray());
                HLAfloat32LE N_LPriorityEncoder =
_encoderFactory.createHLAfloat32LE(Float.parseFloat(manager.jsonReader.getSimulationParameter(federateName,
"N_LPriority")));
                attributeValues.put(_attributeN_LPriority, N_LPriorityEncoder.toByteArray());
                HLAunicodeString F_HPriorityEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "F_HPriority"));
                attributeValues.put(_attributeF_HPriority, F_HPriorityEncoder.toByteArray());
                HLAunicodeString F_LPriorityEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "F_LPriority"));
                attributeValues.put(_attributeF_LPriority, F_LPriorityEncoder.toByteArray());
                HLAunicodeString D_LoadingEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Loading"));
                attributeValues.put(_attributeD_Loading, D_LoadingEncoder.toByteArray());
                HLAunicodeString D_TREncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_TR"));
                attributeValues.put(_attributeD_TR, D_TREncoder.toByteArray());
                HLAunicodeString D_VOLEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_VOL"));
                attributeValues.put(_attributeD_VOL, D_VOLEncoder.toByteArray());
                HLAunicodeString D_TR2Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_TR2"));
                attributeValues.put(_attributeD_TR2, D_TR2Encoder.toByteArray());
                HLAunicodeString D_DepalEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Depal"));
                attributeValues.put(_attributeD_Depal, D_DepalEncoder.toByteArray());

                _rtiAmbassador.updateAttributeValues(ObjectScenario_ApproHandler, attributeValues, null);
            }
            if(federateName.equals("Scenario_Deploy"))
            {
                AttributeHandleValueMap attributeValues =
_rtiAmbassador.getAttributeHandleValueMapFactory().create(1);

                HLAunicodeString FederateNameEncoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "FederateName"));
                attributeValues.put(_attributeScenario_DeployFederateName,
FederateNameEncoder.toByteArray());
                HLAfloat32LE SimTimeEncoder =
_encoderFactory.createHLAfloat32LE(Float.parseFloat(manager.jsonReader.getSimulationParameter(federateName,
"SimTime")));
                attributeValues.put(_attributeScenario_DeploySimTime, SimTimeEncoder.toByteArray());

                HLAunicodeString D_Soudure1Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Soudure1"));
                attributeValues.put(_attributeD_Soudure1, D_Soudure1Encoder.toByteArray());

                HLAunicodeString D_Soudure2Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Soudure2"));
                attributeValues.put(_attributeD_Soudure2, D_Soudure2Encoder.toByteArray());

                HLAunicodeString D_Portique1Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Portique1"));
                attributeValues.put(_attributeD_Portique1, D_Portique1Encoder.toByteArray());

                HLAunicodeString D_Portique2Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Portique2"));
                attributeValues.put(_attributeD_Portique2, D_Portique2Encoder.toByteArray());

                HLAunicodeString D_Colle1Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Colle1"));
                attributeValues.put(_attributeD_Colle1, D_Colle1Encoder.toByteArray());

                HLAunicodeString D_Colle2Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Colle2"));
                attributeValues.put(_attributeD_Colle2, D_Colle2Encoder.toByteArray());

                HLAunicodeString D_Mirror1Encoder =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Mirror1"));
                attributeValues.put(_attributeD_Mirror1, D_Mirror1Encoder.toByteArray());
```

```
                                        HLAunicodeString                    D_Mirror2Encoder                    =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Mirror2"));
                                        attributeValues.put(_attributeD_Mirror2, D_Mirror2Encoder.toByteArray());

                                        HLAunicodeString                    D_AssemblageEncoder                 =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Assemblage"));
                                        attributeValues.put(_attributeD_Assemblage, D_AssemblageEncoder.toByteArray());

                                        HLAunicodeString                    D_testEncoder                       =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_test"));
                                        attributeValues.put(_attributeD_test, D_testEncoder.toByteArray());
                                        HLAunicodeString                    D_PoseEncoder                       =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_Pose"));
                                        attributeValues.put(_attributeD_Pose, D_PoseEncoder.toByteArray());
                                        HLAunicodeString                    D_installEncoder                    =
_encoderFactory.createHLAunicodeString(manager.jsonReader.getSimulationParameter(federateName, "D_install"));
                                        attributeValues.put(_attributeD_install, D_installEncoder.toByteArray());

                                        _rtiAmbassador.updateAttributeValues(ObjectScenario_DeployHandler,      attributeValues,
null);
                        }
                        //Launch simulation
                        ParameterHandleValueMap                             parameters                          =
_rtiAmbassador.getParameterHandleValueMapFactory().create(1);
                        HLAunicodeString IdEncoder = _encoderFactory.createHLAunicodeString();
                        IdEncoder.setValue("Master");
                        parameters.put(_parameterId, IdEncoder.toByteArray());
                        HLAunicodeString DstEncoder = _encoderFactory.createHLAunicodeString();
                        DstEncoder.setValue(federateName);
                        parameters.put(_parameterDst, DstEncoder.toByteArray());
                        HLAunicodeString MsgEncoder = _encoderFactory.createHLAunicodeString();
                        MsgEncoder.setValue("Start");
                        parameters.put(_parameterMsg, MsgEncoder.toByteArray());
                        HLAunicodeString ConfirmEncoder = _encoderFactory.createHLAunicodeString();
                        ConfirmEncoder.setValue("");
                        parameters.put(_parameterConfirm, ConfirmEncoder.toByteArray());

                        _rtiAmbassador.sendInteraction(InteractionMessage, parameters, null);

                        //wait for response
                        if(federateName.equals("Scenario_Appro")){
                                while(Double.isNaN(LoopScenario_Appro)) Thread.sleep(1000);
                                return LoopScenario_Appro;
                        }else if(federateName.equals("Scenario_Deploy")){
                                while(Double.isNaN(LoopScenario_Deploy)) Thread.sleep(1000);
                                return LoopScenario_Deploy;
                        }

                }catch (Exception e)
                {
                        e.getMessage();
                }
                return Double.NaN;
        }

        @Override
        public final void objectInstanceNameReservationSucceeded(String objectName)
        {
         synchronized (_reservationSemaphore) {
          _reservationComplete = true;
          _reservationSucceeded = true;
          _reservationSemaphore.notifyAll();
         }
        }

        @Override
        public final void objectInstanceNameReservationFailed(String objectName)
        {
         synchronized (_reservationSemaphore) {
          _reservationComplete = true;
          _reservationSucceeded = false;
          _reservationSemaphore.notifyAll();
         }
        }
```

```
@Override
public void removeObjectInstance(ObjectInstanceHandle theObject,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        SupplementalRemoveInfo removeInfo)
{
}

@Override
public void reflectAttributeValues(ObjectInstanceHandle theObject,
         AttributeHandleValueMap theAttributes,
         byte[] userSuppliedTag,
         OrderType sentOrdering,
         TransportationTypeHandle theTransport,
         SupplementalReflectInfo reflectInfo)
{
                try{
                if(theObject.equals(ObjectKPIHandler))
                {
                                HLAunicodeString FederateNameDecoder = _encoderFactory.createHLAunicodeString();
                                FederateNameDecoder.decode(theAttributes.get(_attributeKPIFederateName));
                                String FederateName = FederateNameDecoder.getValue();

                                HLAfloat32LE SimTimeDecoder = _encoderFactory.createHLAfloat32LE();
                                SimTimeDecoder.decode(theAttributes.get(_attributeKPISimTime));
                                float SimTime = SimTimeDecoder.getValue();

                                HLAinteger32LE LEADTIMEDecoder = _encoderFactory.createHLAinteger32LE();
                                LEADTIMEDecoder.decode(theAttributes.get(_attributeLEADTIME));
                                int LEADTIME = LEADTIMEDecoder.getValue();

                                HLAinteger32LE WIPDecoder = _encoderFactory.createHLAinteger32LE();
                                WIPDecoder.decode(theAttributes.get(_attributeWIP));
                                int WIP = WIPDecoder.getValue();

                                System.out.println("[" + FederateName + "] [SimTime:" + SimTime + "] [LEADTIME:" + LEADTIME + "] [
WIP:"+WIP+"]");
                }

                } catch (Exception e)
                {
                                e.printStackTrace();
                }
 }
}
```