Université Assane Seck De Ziguinchor (U.A.S.Z)

UFR Sciences Économiques et Sociales (S.E.S)

Département Économie Gestion (ECOGES)

Mater en Management des Systèmes d'Information Automatisés (M.S.I.A)

MÉMOIRE DE FIN D'ÉTUDES DE MASTER

SUJET : « VERS UNE PLATEFORME À BASE D'ONTOLOGIES POUR L'INTÉGRATION DE MODÈLES DE SIMULATION DES MALADIES INFECTIEUSES »

Soutenu le 23/12/2023 par :

M. Mouhamadou Lamine FAYE

Sous l'encadrement et la supervision respectives de :

Dr Papa Alioune CISSE	*Enseignant-chercheur (MCF assimilé)*	*Université Assane Seck de Ziguinchor*
Pr Melyan MENDY	*Enseignant-chercheur (Pr assimilé)*	*Université Assane Seck de Ziguinchor*

Ont siégé dans ce jury :

Pr Melyan MENDY	*Professeur Assimilé*	*Président du Jury*	*UASZ*
Pr Ibrahima DIOP	*Professeur Assimilé*	*Rapporteur*	*UASZ*
Dr Gorgoumack SAMBE	*Maître de conférence Titulaire*	*Examinateur*	*UASZ*
Dr Papa Alioune CISSE	*Maître de conférence Titulaire*	*Encadrant*	*UASZ*

Année Académique 2022-2023

DÉDICACE

A mon très cher père Ismaïla Faye.

A ma très chère mère Ndogou Gueye.

A ma défunte tante Marie Gueye, mon défunt grand-père Idrissa Fall, ma défunte grand-mère Aminata Sène et ma très chère tante Ramatoulaye.

A mon défunt oncle El hadji Youssoupha Ndoye.

A ma très chère tante Salimata Mar.

A mes très chères sœurs, Khadidia, Aminata et Awa Faye.

A mes oncles Malick Gueye, Moussa Faye et Feu Babacar Fall.

A mon cousin, mon frère Abdoulaye Gueye.

A toute la famille Gueye.

A toute la famille Ndoye.

A toute la famille Faye.

A ma chère Fabienne Eliane Malou.

A ma chère Bassine Ba.

A mes camarades de promotion, particulièrement à David Jeremy Daniel Dieme, Maguette Aw, Moussa Namory Diawara et Mame Diarra Mbengue.

A tous mes colocataires.

A mes voisins du quartier Diabir de Ziguinchor.

A mes chers amis.

Je vous dédie ce modeste travail.

REMERCIMENTS

C'est avec plaisir que je réserve ces quelques lignes en signe de gratitude et de profonde reconnaissance à tous ceux qui, de près ou de loin, ont contribué à l'aboutissement de ce travail.

Je souhaite adresser mes remerciements au corps professoral et administratif de l'université Assane Seck de Ziguinchor. Leurs connaissances approfondies, leur dévouement et leur soutien ont joué un rôle déterminant dans mon développement académique.

Mes remerciements vont aussi à l'endroit du corps professoral et administratif de l'université Iba Der Thiam de Thiès (UIDT) où j'ai acquis les fondations solides qui m'ont guidé vers ce prestigieux temple du savoir qu'est l'université Assane Seck de Ziguinchor.

Au degré de cette œuvre, je révèle ma gratitude à mon très cher Professeur et encadrant Monsieur Papa Alioune CISSE qui a assuré l'encadrement du présent travail. Ses critiques et avis ont profondément contribué à bien élaborer le fond et la forme de ce projet de recherche.

Je remercie le Professeur Melyan MENDY, de l'Université Assane Seck de Ziguinchor pour l'honneur qu'il me fait de présider ce jury.

Je remercie le Professeur Ibrahima DIOP, de l'Université Assane Seck de Ziguinchor, d'avoir accepté d'être rapporteur de ce mémoire et membre du jury.

Je remercie également Monsieur Gorgoumack SAMBE, Maître de Conférence Assimilé à l'Université Assane Seck de Ziguinchor, d'avoir accepté d'être examinateur de ce mémoire et membre du jury.

RÉSUMÉ

Le contrôle des maladies infectieuses constitue un défi perpétuel en santé publique. La nécessité de comprendre leur dynamique complexe et l'impératif d'élaborer des stratégies de veille épidémiologique, de contrôle et d'atténuation requièrent souvent des activités de modélisation et de simulation numérique. Cependant, il convient de noter que la modélisation et la simulation de tels systèmes complexes requièrent des compétences expertes sur les outils de modélisation et les plateformes de simulation. Ce qui constitue souvent un blocage pour les praticiens et les débutants dans le domaine. En outre, la communication entre les modélisateurs et les experts du domaine peut être compliquée en raison des différences dans les langages et les concepts utilisés de part et d'autre.

Dans ce mémoire, nous présentons un projet visant à apporter des solutions à ces complications afin de faciliter l'exercice même de modélisation et de simulation des maladies infectieuses, simplifier le processus de simulation des modèles, favoriser la collaboration entre les chercheurs (praticiens et « modélisateurs ») et permettre la réutilisation de modèles et des parties de modèles.

La première étape dans la mise en œuvre de ce projet consiste à proposer une plateforme à base d'ontologies pour automatiser le processus de simulation des modèles de maladies infectieuses. La plateforme comprend un entrepôt de modèles de simulation, une ontologie pour annoter ces modèles et un module d'orchestration autonome des simulations qui assure la sélection, la comparaison, et la composition éventuelle de modèles de simulation.

Dans le cadre de ce mémoire, notre contribution principale à ce projet est le développement d'un « module de gestion des simulations » qui assure l'orchestration autonome du processus de simulation d'un modèle avec une plateforme de simulation spécifiée. Ce module permet, à partir d'une « requête de simulation » émise par un utilisateur par exemple, de requérir les données d'entrée du modèle à simuler, d'exécuter les simulations demandées et offrir à l'utilisateur des mécanismes facilitant l'exploitation des résultats des simulations.

Mots-clés: maladies infectieuses, modélisation, simulation numérique, modèles mathématiques à base d'équations, modèles informatiques à base d'agents, ontologies, systèmes de veille épidémiologique, Framework Django.

ABSTRACT

Controlling infectious diseases poses a perpetual challenge in public health. The need to understand their complex dynamics and the imperative to develop epidemiological surveillance, control, and mitigation strategies often require modeling and numerical simulation activities. However, it should be noted that modeling and simulating such complex systems require expert skills in modeling tools and simulation platforms. This often serves as a barrier for practitioners and beginners in the field. Furthermore, communication between modelers and domain experts can be complicated due to differences in languages and concepts used on both sides.

In this thesis, we present a project aimed at providing solutions to these complications to facilitate the modeling and simulation of infectious diseases, simplify the model simulation process, promote collaboration between researchers (practitioners and "modelers"), and enable model and model component reuse.

The first step in implementing this project is to propose an ontology-based platform to automate the simulation process of infectious disease models. The platform includes a simulation model repository, an ontology to annotate these models, and an autonomous simulation orchestration module that ensures the selection, comparison, and possible composition of simulation models.

In the context of this thesis, our main contribution to this project is the development of a "simulation management module" that ensures the autonomous orchestration of the simulation process for a specified simulation platform. This module allows, for example, the issuance of a "simulation request" by a user to request input data for the model to be simulated, execute the requested simulations, and provide the user with mechanisms to facilitate the exploitation of simulation results.

Keywords: infectious diseases, modeling, numerical simulation, equation-based mathematical models, agent-based computer models, ontologies, epidemiological surveillance systems, Django Framework.

LISTE DES TABLEAUX ET DES FIGURES

LES TABLEAUX

LES FIGURES

LISTES DES CIGLES ET ABBREVIATIONS

ODS : Ontology Driven Simulation

SMA : Système Multi-Agents

OMS : Organisation Mondiale de la Santé

IHME: Institute for Health Metrics and Evaluation

ECDC: European Centre for Disease Prevention and Control

WHO: World Health Organization

CDC: Centers for Disease Control and Prevention

VIH : Virus de l'Immunodéficience Humaine

COVID-19 : COronaVIrus Disease of 2019

GLEAMviz: Global Epidemic and Mobility Modeler

EMOD: Epidemic Modeling

UML: Unified Modeling Language

SI: Système d'Informations

HOOD: Hierarchical Object Oriented Design

OMT: Object Modeling Technique

OOA: Object Oriented Analysis

OOD: Object Oriented Design

OOM: Object Oriented Merise

OOSE: Object Oriented Software Engineering

OMG: Object Management Group

EDO: Équations Différentielles Ordinaires

GAMA: Geographical Information Systems Agent-based Modeling Architecture

GAML: GAma Modeling Language

XML: Extensible Markup Language

HTML: HyperText Markup Language

MVC : Modèle-Vue-Contrôleur

URL : Uniform Resource Locator

ORM: Object Relational Mapping

SQL: Structured Query Language

ACID : Atomicité, Cohérence, Isolation, Durabilité

VEMIS : Veille Épidémiologique des Maladies Infectieuses au Sénégal

TABLE DES MATIÈRES

Introduction générale

Nous abordons dès l'introduction trois thématiques dans lesquelles se situent nos travaux: les systèmes de veille épidémiologique, l'épidémiologie informatique (ou "Computational epidemiology") et les simulations dirigées par une ontologie (ou "Ontology Driven Simulation" – ODS).

Les systèmes de veille épidémiologique sont utilisés pour contrôler la propagation des maladies en proposant des plans d'actions visant à prévenir les risques identifiés (Camara, 2013). L'intérêt d'un système de veille épidémiologique est de permettre et de faciliter la prédiction des risques et la prise de décision en se fondant sur des analyses quantitatives, réalisées sur la base de simulations de modèles numériques. Ces modèles, construits à l'issue d'études épidémiologiques, permettent d'expliquer la dynamique de propagation des maladies et de valider des hypothèses.

Le "Computational epidemiology" est une discipline dont l'objectif principal est l'application des concepts et des ressources informatiques (notamment les techniques, approches et outils de modélisation et de simulation) et géographiques (notamment les outils de représentation et de visualisation des données géospatiales complexes) pour fournir aux épidémiologistes des outils conviviaux, afin de leur permettre de mieux comprendre les problèmes fondamentaux de l'épidémiologie, tels que la propagation des maladies, l'efficacité d'une intervention de santé publique, la prédiction et l'analyse des manifestations des maladies et de leur propagation dans une population donnée, … (Cissé, 2016), (Cisse et al., 2017).

ODS désigne un processus qui utilise les connaissances codées dans les ontologies[1] pour concevoir de manière dynamique et automatique des modèles de simulation. Il s'agit d'avoir d'une part les ontologies du domaine ou de l'application et d'autre part les ontologies de modélisation (codage des informations sur la modélisation comme les composants des modèles, les différentes phases et activités de modélisation, etc.). Ensuite, les concepts d'ontologies de domaine sont mappés aux concepts d'ontologies de modélisation, puis des instances d'ontologies de modélisation sont créées pour représenter un modèle. Une fois les instances

[1] Une définition plus détaillée du domaine des ontologies est donnée dans la section 2.10 du chapitre 1. Nous proposons ici une définition informatique simple pour comprendre cet énoncé : une ontologie est la représentation formelle d'un champ d'informations par les termes et concepts liés.

d'ontologie représentant le modèle créées, des outils supplémentaires sont utilisés pour les traduire en modèles de simulation exécutables (Cisse et al., 2019).

La plateforme que nous présentons dans ce mémoire tire ses racines de ces trois thématiques. En effet, il s'agit d'une plateforme à base d'ontologies pour automatiser le processus de simulation des modèles des maladies infectieuses, qui orchestre de façon autonome le processus de simulation en guidant la sélection, la comparaison et la composition (appelée aussi couplage) éventuelle des modèles de simulation.

Sur le plan des systèmes de veille épidémiologique, la plateforme offre les services d'un moteur de simulation de modèles numériques qui dispose d'un important "entrepôt" de modèles de simulation sur les maladies infectieuses (on l'appelle aussi "*modèlethèque*" ou bibliothèque de modèles). Elle est invoquée, en fournissant une "*requête de simulation*", par :

- Un utilisateur humain qui dispose de données numériques ou d'interrogations sur une maladie particulière ;
- Ou par un quelconque dispositif de veille épidémiologique qui collecte des données épidémiologiques de terrains.

Sur le plan "Computational epidemiology", la plateforme intègre, autant que possible, de récentes solutions issues des avancées dans la recherche en informatique permettant de relever les nouveaux défis épidémiologiques contemporains de modélisation et de simulation de la propagation des infectieuses. Il s'agit, entre autres :

- De l'utilisation des Systèmes Multi-Agents (SMA) comme approche de modélisation et de simulation pour saisir la complexité inhérente aux propagations des maladies infectieuses qui est, en parties, impliquée dans les interactions et les comportements humains qui, sont appréhendés à travers les réseaux d'interactions sociaux et spatiaux.
- De l'utilisation du Machine Learning pour intégrer des données (qui sont parfois rares, largement distribuées et incomplètes) dans les modèles de simulation.
- De l'utilisation de nouvelles techniques de représentation et de visualisation graphique (par exemple, les Systèmes d'Information Géographique et la réalité virtuelle) des données d'entrées et de sorties de simulations.

Sur le plan ODS, la plateforme intègre une ontologie de domaine pour l'annotation sémantique des modèles de simulation, des données et des ressources de simulation. Sur la base de cette ontologie et de la "*modèle-thèque*", la plateforme permet une orchestration du processus de

simulation en guidant la comparaison, la sélection et la composition des modèles de simulation numériques pour répondre à une "*requête de simulation*".

Dans le cadre de nos travauxde mémoire, nous nous concentrerons sur le développement du module "gestionnaire de simulation", qui constitue le cœur de notre système. Ce module permettra aux utilisateurs de spécifier les paramètres d'entrée des modèles, de lancer des simulations et d'analyser les résultats. De plus, nous initialiserons les modules complémentaires tels que le "gestionnaire des requêtes et sorties de simulations", les "interfaces utilisateurs" et la "base de données", assurant ainsi la cohérence et le stockage des données.

Ce document est réparti en trois principaux chapitres:

- Le premier chapitre, intitulé « généralités du projet et cadrage de nos travaux », présente les généralités du projet dans le cadre duquel se situent nos travaux de mémoire. Nous proposons dans un premier temps quelques définitions de notions essentielles qui permettront de faciliter la lecture du document. Ensuite, nous présentons un état de l'art sur notre thème de recherche. Puis, nous présentons les problématiques qui sous-tendent le projet, ses objectifs et les résultats attendus. Enfin, nous présentons l'architecture générale de la plateforme qui est en cours de développement dans le cadre de ce projet. Ce qui nous permet par la suite de faire un cadrage fixant les limites de nos travaux de mémoire dans le projet global.
- Le deuxième chapitre intitulé « Analyse et conception » donne un aperçu sur UML qui est l'outil d'analyse et de conception utilisé dans ce travail. Ensuite, nous présentons de façon détaillée le problème spécifique abordé dans ce mémoire. Ensuite, nous faisons une analyse de la solution que nous avons proposée. Cette analyse consiste à faire l'inventaire des fonctionnalités de notre solution. Cela inclût les diagrammes de cas d'utilisation montrant les relations entre les acteurs du système et leurs besoins fonctionnels. Enfin, nous y aborderons la conception de notre solution qui sera illustrée par un dictionnaire des données et un diagramme de classes pour l'implémentation de la plateforme.
- Le troisième et dernier chapitre décrit, d'abord, les technologies utilisées pour implémenter notre solution. Ensuite, nous présentons les détails d'implémentation de la solution. Enfin, nous présentons le module « gestionnaire des simulations » développé, ses fonctionnalités et ses interfaces.

- Enfin, nous terminerons par une conclusion générale, sans oublier d'évoquer d'éventuelles perspectives.

Chapitre 1. Généralités du projet et cadrage de nos travaux

1.1. Définitions

1.1.1. Système complexe

Le comportement global d'un système, constitué d'entités multiples et éventuellement hétérogènes, s'explique par le résultat du jeu d'interactions entre ses entités constitutives. C'est la possibilité de pouvoir déterminer le comportement global du système qui permet de le caractériser de *simple*, et dans le cas contraire, de *complexe* (Treuil et al., 2008). Un système complexe ne révèle pas un déterminisme latent qu'il est possible de calculer mais manifeste une certaine forme d'imprévisibilité possible et d'émergence plausible du nouveau (Le Moigne, 1990). En effet, les entités qui composent un système complexe sont dotés de comportements et parfois de mécanismes de décisions qui orientent leurs comportements indépendamment des autres entités. C'est l'interaction et la dynamique d'interaction entre ces entités qui déterminent le comportement global du système.

Cependant, l'imprévisibilité des systèmes complexes n'exclut pas leur intelligibilité. Ne pouvant les représenter de manière définitive, c'est-à-dire, produire des représentations finies et statiquement prêtes à être utilisées, on peut, en un moment donné, y faire des représentations elles-mêmes complexes pour pouvoir permettre un raisonnement. Ce procédé qui consiste à produire une représentation simplifiée d'un système pour le rendre intelligible est appelé *modélisation* et son résultat *modèle*.

1.1.2. Modélisation

La modélisation est l'activité qui consiste à construire des modèles. C'est l'une des deux principales composantes, avec l'expérimentation (on dira "simulation"), de la démarche scientifique (Treuil et al., 2008). En effet, la modélisation et la simulation sont deux activités qui vont de pair et qui forment ensemble une approche scientifique à part entière dont le but est de proposer des théories, outils et vocabulaires spécifiques afin de produire des connaissances sur des phénomènes naturels et artificiels (Siebert, 2011).que les grandeurs caractéristiques

L'activité de modélisation consiste à construire, à partir d'un système ou d'un phénomène que l'on étudie, une abstraction de celui-ci (le modèle), qui ne retient du système que les grandeurs caractéristiques (les variables d'états) jugées pertinentes par le modélisateur (Fianyo, 2001).

1.1.3. Modèle

Un modèle, en science, est une image stylisée et abstraite d'une portion de réalité (Ferber, 1995). Autrement dit, un modèle est une représentation simplifiée d'un système selon un point de vue ou une question bien déterminée.

L'activité scientifique consiste principalement à faire des modèles des phénomènes et des objets qu'elle étudie (Ferber, 1995). Il convient tout d'abord de séparer modèles "physiques" et modèles "abstraits". Les premiers, qui regroupent les maquettes, les modèles réduits ou les modèles animaux, sont des dispositifs du monde réel conçus pour être soumis à expérimentation. Les seconds concernent les modèles conçus pour être implémentés et exécutés/simulés sur ordinateur (Treuil et al., 2008). C'est cette deuxième catégorie de modèles qui nous intéresse dans le cadre de ce travail.

L'intérêt d'un modèle est d'abord d'être plus explicite, plus simple et plus facile à manipuler que la réalité qu'il est censé représenter. Les modèles éliminent ainsi un grand nombre de détails considérés comme inutiles par le modélisateur afin de mieux se consacrer aux données que celui-ci juge pertinentes relativement au problème qu'il désire résoudre. Les modèles sont ainsi des images homomorphes de la réalité, c'est-à-dire qu'il existe un homomorphisme entre l'objet d'étude et le modèle qui permet d'appliquer les résultats du modèle à l'objet lui-même, comme le montre la figure suivante (Ferber, 1995).

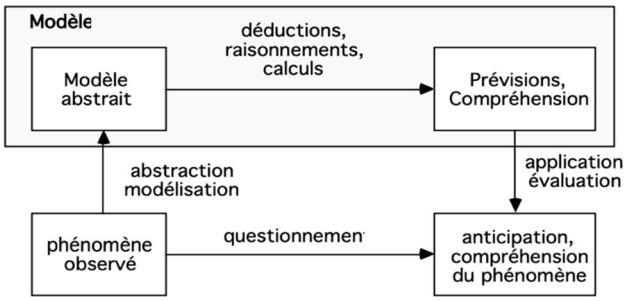

Figure 1. La prévision et la compréhension de phénomènes passent par l'élaboration de modèles (Ferber, 1995).

Le phénomène observé est traduit sous la forme d'une abstraction, laquelle peut être manipulée (par simulations par exemple) pour obtenir des résultats qui peuvent alors servir à mieux comprendre ou à prédire des situations futures. C'est par des opérations de cette nature que l'on

parvient à calculer les orbites des planètes autour du Soleil, à prédire l'évolution d'une maladie dans une population donnée, à évaluer des politiques de contrôle (vaccins, traitements, …) de la propagation d'une maladie ou d'une épidémie, etc.

1.1.4. Modèle statique vs modèle dynamique

Un modèle est appelé statique quand il a pour but de représenter la structure d'un système de référence photographié à un instant donné, sans allusion à son évolution dans le temps. Inversement, un modèle sera appelé dynamique quand il inclura dans sa représentation des hypothèses ou des règles concernant l'évolution dans le temps du système de référence (Treuil et al., 2008). Il convient de retenir donc que seuls les modèles dynamiques sont susceptibles d'être soumis à un processus d'expérimentation ou de simulation. Un modèle statique, par définition, ne peut pas être simulé.

1.1.5. Simulation

La simulation est l'activité par laquelle, en fonction d'objectifs précis, et à l'aide d'un dispositif expérimental informatique (appelé simulateur), on perturbe selon un protocole déterminé un modèle dynamique, en faisant évoluer ses entrées et en récupérant les sorties (Treuil et al., 2008).

1.1.6. Simulateur / plateforme de simulation

Une plateforme de simulation est un programme informatique permettant de simuler un phénomène réel sur ordinateur. Autrement dit, un simulateur permet la reproduction d'un environnement ou d'un processus de manière virtuelle.

1.1.7. Paramètres d'entrée / de sortie d'un modèle

Les entrées d'un modèle dynamique sont des paramètres dont la valeur est définie en dehors du modèle et qui représentent ce que le simulateur peut perturber. Les sorties d'un modèle dynamique sont également des paramètres qui expriment ce que l'on cherche à mesurer en réponse à ces perturbations (Treuil et al., 2008).

1.1.8. Maladie infectieuse

Une maladie infectieuse est une maladie causée par la propagation d'un agent infectieux, tel qu'un virus, une bactérie, un parasite ou un champignon. Ces agents infectieux pénètrent dans

le corps d'un individu et se multiplient, ce qui peut entraîner des symptômes et des complications de santé (Cuzin and Delpierre, 2005).

Les maladies infectieuses peuvent être transmises de différentes manières, notamment par contact direct avec une personne infectée, par des gouttelettes respiratoires émises lors de la toux ou de l'éternuement, par la consommation d'aliments ou d'eau contaminés, par des piqûres d'insectes vecteurs, ou encore par des contacts avec des objets contaminés (Fontenille et al., 2013).

1.1.9. Épidémiologie

L'épidémiologie se définit, dans les recommandations « déontologie et bonnes pratiques en épidémiologie » de 1998 de l'OMS, comme étant « *une discipline scientifique qui étudie notamment les différents facteurs intervenant dans l'apparition des maladies ou de phénomènes de santé ainsi que leur fréquence, leur mode de distribution, leur évolution et la mise en œuvre des moyens nécessaires à leur prévention* ». Les objectifs principaux de l'épidémiologie sont la promotion de la santé et la réduction des problèmes de santé.

1.1.10. Ontologie

L'ontologie désigne une représentation formelle et explicite des concepts et des relations existant dans un domaine particulier. La première définition de l'ontologie informatique fut apportée par Robert Neches qui fut chercheur en technologies de l'information à l'université de Californie du sud. Dans le cadre de l'intelligence artificielle, R. Neches et ses collègues publient un article intitulé « Enabling technology for knowledge sharing » où ils proposent la définition suivante : « Une ontologie définit les termes et les relations de base du vocabulaire d'un domaine ainsi que les règles qui indiquent comment combiner les termes et les relations de façon à pouvoir étendre le vocabulaire (Fernandez, n.d.)».

Une ontologie informatique définit les termes, les concepts et les relations utilisés pour décrire et représenter les connaissances d'un domaine bien déterminé. Les ontologies permettent de clarifier la signification des termes utilisés dans un domaine, d'identifier les relations entre les concepts, d'établir des hiérarchies et des taxonomies, et de créer des structures de connaissances plus riches et plus interconnectées. Elles favorisent également l'interopérabilité entre les systèmes d'information en fournissant un cadre commun de compréhension des données.

1.2. État de l'art

La revue de la littérature que nous avons effectuée nous a permis de découvrir quelques travaux allant dans le sens de l'étude que nous proposons dans ce projet, notamment dans l'entreposage de modèles de maladies infectieuses. En effet, il existe plusieurs bases de données, ressources en ligne et initiatives de modélisation de centres de recherche, d'universités et d'organisations internationales :

- Institute for Health Metrics and Evaluation (IHME) : IHME fournit des modèles de prévision pour différentes maladies infectieuses, notamment des prévisions de morbidité, de mortalité et de prévalence("Homepage | The Institute for Health Metrics and Evaluation," n.d.) (Europe, 2019).
- Johns Hopkins University Coronavirus Resource Center("COVID-19 Map," n.d.) qui a participé à la lutte contre la pandémie de COVID-19 en fournissant des modèles et des statistiques.
- European Centre for Disease Prevention and Control (ECDC) ("What we do," 2010) : L'ECDC propose des modèles de maladies infectieuses, des données épidémiologiques et des rapports sur les maladies infectieuses en Europe.
- World Health Organization (WHO) : L'OMS publie des rapports, des statistiques et des modèles sur diverses maladies, y comprises celles infectieuses, en mettant l'accent sur la santé mondiale(Carlson et al., 2023).
- Centers for Disease Control and Prevention (CDC) : Le CDC américain propose des données, des modèles et des ressources sur les maladies infectieuses, y compris la grippe, le VIH(Workowski, 2015), etc.
- Modélisation informatique en open-source : Il existe des projets open-source tel que COVID-19 Scenario Modeling Hub(Howerton et al., 2023) qui regroupe de multiples modèles de pandémie, et particulièrement des modèles spécifiques à la COVID-19 permettant d'avoir des résultats fiables et synthétiques pour de meilleures projections.

Par ailleurs, nous avons aussi répertorié plusieurs plateformes dédiées à la simulation de modèles des maladies infectieuses qui permettent aux chercheurs, aux professionnels de la santé et aux décideurs politiques de créer, d'exécuter et d'analyser des simulations informatiques pour mieux comprendre la propagation des maladies infectieuses et évaluer l'impact de différentes interventions. Ces plateformes varient en termes de fonctionnalités, de complexité et d'objectifs

de modélisation. Elles sont souvent utilisées par des chercheurs, des épidémiologistes et des décideurs pour mieux comprendre les dynamiques de propagation des maladies infectieuses et évaluer l'efficacité des stratégies d'intervention.

Nous pouvons citer quelques exemples de plateformes à savoir:

- Les outils les plus populaires restent MATLAB et ArcGIS. Une combinaison des deux outils est souvent faite pour des rendus plus présentables de résultats. MATLAB est souvent utilisé pour développer des modèles mathématiques de propagation de maladies infectieuses et même de les simuler. Tandis que le logiciel ArcGIS aide à visualiser et analyser les données spatiales liées à ces modèles développés.
- GLEAMviz : GLEAMviz (Global Epidemic and Mobility Modeler) est une plateforme de modélisation qui intègre des données de mobilité humaine et des facteurs épidémiologiques pour simuler la propagation des maladies infectieuses à l'échelle mondiale.
- EpiModel : EpiModel est une plateforme open-source qui facilite la création, l'exécution et l'analyse de modèles épidémiologiques complexes pour différentes maladies infectieuses.
- EMOD (Epidemic Modeling) : EMOD est une plateforme développée par le Naval Research Laboratory qui permet de simuler la propagation de maladies infectieuses dans des populations humaines. Elle est souvent utilisée pour étudier la transmission du paludisme, de la dengue et d'autres maladies.
- Agent-Based Modeling Platforms : Certaines plateformes de modélisation à base d'agents, telles que NetLogo et AnyLogic, peuvent être utilisées pour créer des modèles de maladies infectieuses et simuler leur propagation en prenant en compte le comportement individuel des agents.

L'utilisation de plateformes pour la simulation et l'entreposage de modèles de maladies infectieuses peut offrir plusieurs avantages. Ces plateformes peuvent aider les agents de santé et les décideurs à évaluer diverses mesures de lutte contre les maladies infectieuses, telles que la vaccination, la fermeture des écoles et l'isolement. La simulation permet de tester différentes stratégies d'atténuation et de prendre en compte la variation et l'incertitude dans les paramètres de la maladie, ainsi que l'aléatoire dans la progression d'une épidémie. L'entreposage de modèles peut également faciliter la collaboration entre les chercheurs et les praticiens, en permettant le partage de données et de connaissances. En somme, l'utilisation de plateformes

pour la simulation et l'entreposage de modèles de maladies infectieuses peut améliorer notre compréhension des maladies infectieuses et notre capacité à y faire face.

1.3. Présentation du projet

1.3.1. Problématiques

La propagation des maladies infectieuses fait intervenir un très grand nombre d'entités (hôte, vecteur de transmission, agent pathogène, facteurs de risque, etc.) dont les interactions donnent lieu à l'émergence d'évènements (nouveaux cas, infestation d'un point d'eau, etc.) pouvant se situer à différents niveaux d'échelles spatiales (régionale, continentale, mondiale) et temporelles (saisonnière par exemple). Ces types de phénomènes, par leurs évolutions et leurs émergences issues des interactions des éléments les composant, sont qualifiés de systèmes complexes (Camara, 2013), (Cissé, 2016). Pour analyser l'évolution et la dynamique de tels systèmes complexes, il est indispensable de recourir à la modélisation systémique (Le Moigne, 1990) qui consiste à construire un modèle reproduisant son comportement dans le cadre d'une simulation.

Cependant, l'exercice de modélisation et de simulation de systèmes complexes en général et des maladies infectieuses en particulier, requièrent des connaissances expertes sur les outils de modélisation (en mathématique et/ou en informatique) d'une part et sur les plateformes de simulations d'autre part. Ce qui constitue parfois un blocage important pour les thématiciens (épidémiologistes, biologistes, …).

D'autre part, pour modéliser une maladie ou une partie d'une maladie, l'informaticien ou le mathématicien modélisateur a besoin de connaissances expertes et de données consistantes sur la maladie. Ces données sont parfois très difficiles à trouver chez les thématiciens. En plus, même si les données sont disponibles, la communication entre le modélisateur et l'expert du domaine peut parfois être difficile à cause des écarts qu'il peut y avoir sur les concepts, termes et outils utilisés par les uns et les autres, dû au manque d'un cadre commun de partage et d'échange.

En outre, pour un modélisateur débutant, l'exercice d'apprentissage et d'auto-formation sur la modélisation d'une maladie peut être compliqué par un manque de références et d'exemples de modèles sur lesquels il peut s'appuyer.

1.3.2. Objectif général

L'objectif général du projet est de proposer une plateforme à base d'ontologies pour automatiser le processus de simulation de modèles des maladies infectieuses. La plateforme devra orchestrer de façon autonome le processus de simulation des modèles de maladies infectieuses en guidant la sélection, la comparaison et la composition des modèles de simulation.

1.3.3. Objectifs spécifiques

L'objectif général est incliné en trois objectifs spécifiques :

- **Objectif 1 :** Mettre en place une bibliothèque (entrepôt) de modèles de simulations (modèle-thèque) des maladies infectieuses.
- **Objectif 2 :** Construire une ontologie de domaine des modèles de simulation des maladies infectieuses permettant d'annoter les modèles du « modèle-thèque ».
- **Objectif 3 :** Implémenter la plateforme d'orchestration et d'automatisation des simulations.

1.3.4. Résultats attendus

La plateforme devra permettre entre autres de:

- **Faire des simulations :** un seul modèle à la fois / par composition de modèles (plusieurs modèles / simulateurs).
- **Faciliter la collaboration :** La plateforme devra permettre l'interopérabilité des outils de simulation, la réutilisation des modèles et le partage des données entre les chercheurs dans le domaine de la modélisation des maladies infectieuses. Ce qui permettra d'améliorer les possibilités de collaboration.
- **Servir de support de recherche :** Les articles publiés autour des modèles de simulation des maladies infectieuses peuvent être liés à l'ontologie pour aider les chercheurs à trouver plus rapidement des articles de recherche plus pertinents.
- **Partager des composants de modèles de simulation :** En fournissant une ontologie, une infrastructure est créée pour stocker et récupérer des composants de modèles de simulation exécutables. Ce qui permettra de faciliter l'exercice de modélisation.

1.4. Architecture de la plateforme et cadrage de nos travaux de mémoire

1.4.1. Architecture de la plateforme

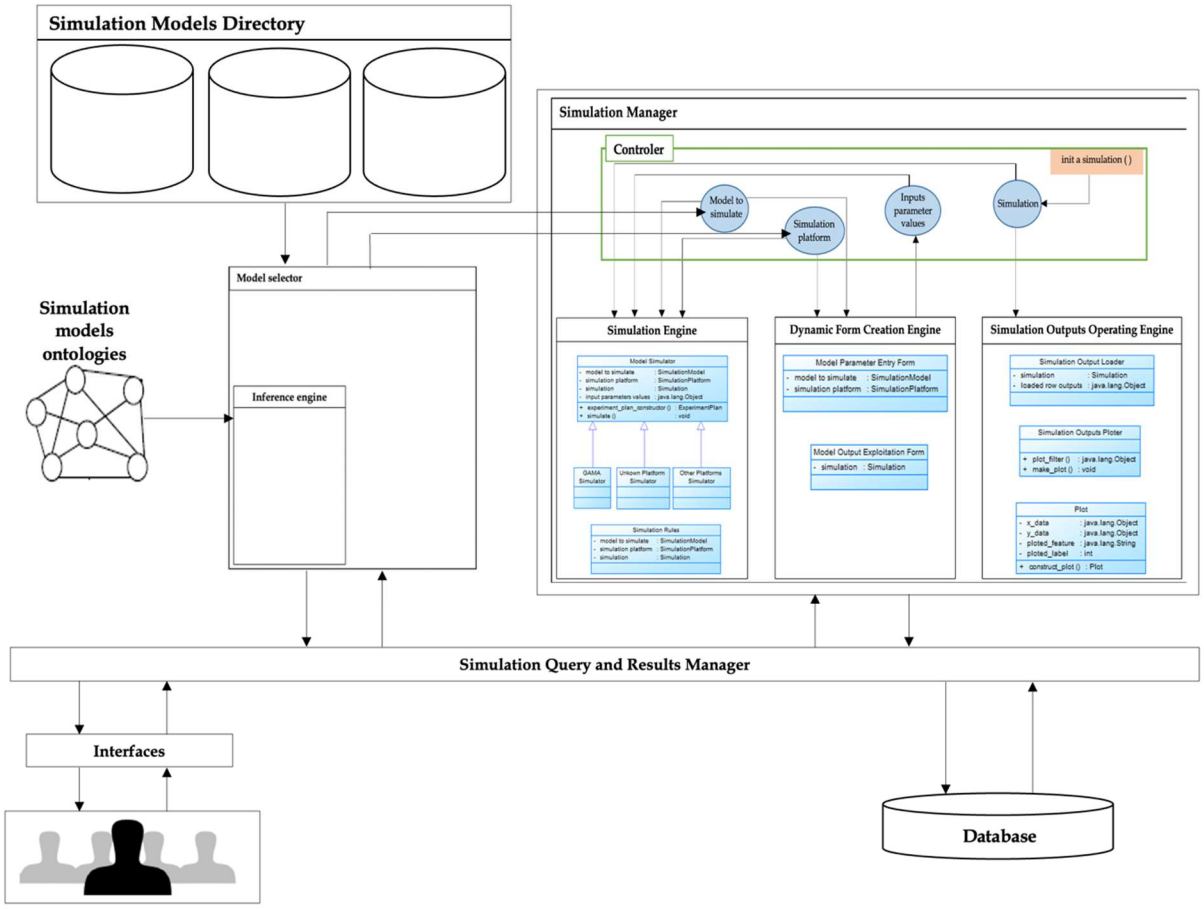

Figure 2. Architecture de la plateforme (première version)

Cette architecture[2] est composée globalement d'un module « sélecteur de modèles », d'un module d'assistance, d'un module « Gestionnaire des simulations », d'une base de données et d'interfaces utilisateur.

1.4.1.1. Le sélecteur de modèles

Ce module appelé « Model selector » ou « sélecteur de modèles » se base sur un entrepôt de modèles de simulation « Simulation models repository » et sur une ontologie des modèles de simulation « Simulation models ontology » des maladies infectieuses accompagnée d'un

[2] Cette première version de l'architecture ne prend pas encore en compte les aspects liés à la composition de modèles.

moteur d'inférence. Il permet, à partir d'une requête utilisateur, d'identifier et sélectionner le ou les modèle(s) de simulation à mettre en œuvre pour satisfaire la requête d'un utilisateur.

1.4.1.2. Le module d'assistance

Comme son nom l'indique, ce module d'assistance, appelé « Simulation queries and results manager » ou « Gestionnaire de requêtes et de sorties de simulation », traite les requêtes et les résultats de simulations des utilisateurs. Il réceptionne les requêtes formulées depuis l'interface utilisateur et interroge le module « model selector » afin de déterminer le(s) modèle(s) à simuler. Il interroge ensuite le module « Simulation manager » pour simuler le(s) modèle(s) en question et récupère les résultats de simulations pour les traiter.

1.4.1.3. L'interface utilisateur

Une « interface utilisateur » permettant aux utilisateurs d'interagir avec le module d'assistance en utilisant des fonctionnalités prédéfinies via des requêtes de simulations. Les requêtes des utilisateurs peuvent être accompagnées de données qui seront fournies en entrée toujours au moyen de cette interface. Les résultats seront fournis également par ce composant.

1.4.1.4. Le gestionnaire des simulations

Le module appelé « Simulation manager » ou « Gestionnaire des simulations » se charge de la mise en œuvre des simulations. Il est composé d'un « moteur de simulation », d'un « moteur de création dynamique de formulaire », d'un « moteur d'exploitation de sorties de simulations » et d'un « contrôleur ».

1.4.1.4.1. Le contrôleur

C'est le point d'entrée du module de « Simulation manager ». Il joue le rôle de chef d'orchestre permettant d'« initialiser » un objet qui représente une simulation, d'orchestrer le déroulement et l'exploitation des résultats d'une simulation. Il prend en entrée un modèle à simuler, une plateforme de simulation, la liste des paramètres d'entrées du modèle à simuler et « initialise » la simulation. Selon les paramètres d'entrée du modèle à simuler, le contrôleur interroge le « moteur de création dynamique de formulaire » pour créer dynamiquement un formulaire permettant à l'utilisateur de donner des valeurs aux paramètres d'entrée du modèle. Ces valeurs d'entrée de simulation, accompagnés du modèle à simuler ; de la plateforme de simulation et de la « simulation » initialisée sont ensuite fournie au « moteur de simulation » pour effectuer

les simulations. Après simulations, l'objet « simulation » est fourni au « moteur d'exploitation des sorties » pour permettre à l'utilisateur d'exploiter ses résultats de simulations.

1.4.1.4.2. Le moteur de création dynamique de formulaire

Le « moteur de création dynamique de formulaire » a pour rôle de créer dynamiquement des formulaires conformément aux besoins de l'utilisateur. Il comporte deux classes permettant créer les formulaires des paramètres d'entrée d'une simulation (« Model parameter entry form ») et les formulaires d'exploitation des sorties de simulation (« Model outputs exploitation form »). En effet, quand un modèle doit être simulé avec des valeurs à fournir par l'utilisateur, un formulaire est dynamiquement créé conformément aux paramètres d'entrée du modèle. Pour exploiter les données brutes de sorties d'une simulation (qui peuvent parfois être nombreuses et complexes), un formulaire peut être créé pour permettre à l'utilisateur de filtrer les données suivant certains critères conformément aux sorties de simulations du modèle.

1.4.1.4.3. Le moteur de simulation

Le « moteur de simulation » permet de mettre en œuvre la simulation d'un modèle dans une plateforme avec des données d'entrée fournies par l'utilisateur. Il comporte une classe abstraite (Model simulator) de simulation de modèles avec une méthode abstraite de (construction de « plans d'expérimentation ») et une méthode (simulate) qui se charge de lancer les simulations après la construction des « plans d'expérimentation ». Il comporte également un ensemble de classes de simulation de modèles dans des plateformes spécifiques (GAMA Simulator, Unknown Platform Simulator, etc.). En effet, pour chaque plateforme de simulation pouvant être utilisée dans la simulation d'un modèle, il faut définir une classe de simulation qui hérite de « Model simulator » et qui implémente sa propre « construction de plans d'expérimentation ». La dernière classe (Simulation Rules) du moteur de simulation permet d'effectuer les expérimentations en fonction de la plateforme de simulation utilisée. C'est cette classe qui sera instanciée à chaque fois qu'une simulation est demandée.

1.4.1.4.4. Le moteur d'exploitation

Le « moteur d'exploitation des résultats de simulation ». Il permet à l'utilisateur d'exploiter les résultats d'une simulation, en visualisant par exemple, via une interface utilisateurs, les données issues des simulations. Les données peuvent être chargées dans leur état bru ou visualisées avec des graphiques.

1.4.2. Cadrage de nos travaux de mémoire

Le domaine de la simulation des maladies infectieuses est d'une importance capitale dans le domaine de la santé publique. La capacité à simuler et à prévoir la propagation des maladies est essentielle pour prendre des décisions éclairées en matière de politique de santé et pour élaborer des stratégies de prévention et de contrôle des épidémies. Cependant, la modélisation de ces phénomènes complexes nécessite des outils sophistiqués et des systèmes informatiques robustes pour gérer efficacement les simulations. Les défis actuels de la modélisation des maladies infectieuses résident dans la nécessité d'automatiser et d'optimiser le processus de simulation. Face à une diversité de modèles et de scénarios, il est crucial de développer une infrastructure permettant une gestion intégrée des simulations, tout en facilitant la collaboration et la réutilisation des modèles.

Tout d'abord, il s'agit de répondre à un besoin croissant d'outils de simulation suffisamment adaptés au contrôle des maladies infectieuses. De plus, l'automatisation des processus de simulation permettra de gagner du temps et de réduire les erreurs humaines, améliorant ainsi la fiabilité des résultats obtenus. En outre, en développant une architecture modulaire et extensible, nous pourrons facilement intégrer de nouveaux modèles de maladies et étendre les fonctionnalités du système à l'avenir.

Nos travaux de mémoire constituent le noyau de ce projet. Les objectifs principaux de notre travail sont les suivants :

- Concevoir et développer le module "gestionnaire de simulation" qui servira de cœur du système, permettant aux utilisateurs de spécifier les paramètres d'entrée des modèles, de lancer des simulations et d'analyser les résultats.
- Initialiser les modules complémentaires tels que le "gestionnaire des requêtes et sorties de simulations" pour gérer efficacement les demandes des utilisateurs et les données de sortie des simulations.
- Développer des interfaces utilisateur pour faciliter l'interaction avec le système, notamment pour la saisie des paramètres d'entrée, la configuration des types de simulations (unitaire ou répétitive) et la visualisation des résultats.
- Mettre en place une base de données pour stocker les modèles de maladies, les résultats de simulation et les données utilisateur, garantissant ainsi la cohérence et la sécurité des données.

Ensemble, ces objectifs contribueront à la réalisation d'une plateforme d'orchestration de simulation, capable de répondre aux besoins de nos utilisateurs.

Par rapport aux objectifs spécifiques de la plateforme (section 3.3 de ce chapitre), il ne s'agit pas pour nous de mettre en place l'entrepôt de modèle de simulation, ni de proposer une ontologie des modèles de simulation des maladies infectieuses. Il s'agit d'intervenir dans la mise en place de la plateforme d'orchestration des simulations.

Par rapport à l'architecture de la plateforme, nos travaux de mémoire consistent à développer le module « gestionnaire de simulation » et initialiser les modules « gestionnaire des requêtes et sorties de simulations », « interfaces utilisateurs » et la « base de données ».

Donc, nos travaux se concentrent sur le développement du module « gestionnaire de simulation ». Ce module aura pour mission de faciliter la requête, la fourniture de données d'entrée, l'exécution des simulations, et l'exploitation des résultats. En développant le module « gestionnaire de simulation » et initialisant ses composantes de base associées, notre travail offre une perspective innovante pour l'avenir de la simulation des maladies infectieuses. C'est à partir de là que d'autres modules pourront se coiffer à l'existant.

Pour atteindre ces objectifs, nous adopterons une approche méthodique basée sur les principes du génie logiciel. Nous suivrons les étapes classiques du cycle de développement logiciel, notamment l'analyse des besoins, la conception du système, l'implémentation des modules, les tests et la validation. De plus, nous mettrons l'accent sur la modularité, la maintenabilité et la scalabilité du système, en utilisant des technologies modernes et des bonnes pratiques de programmation.

Chapitre 2. Analyse et conception

2.1. Aperçu sur UML

2.1.1. Définitions et historique

UML (Unified Modeling Language ou Langage de Modélisation Unifié en français) est un langage visuel utilisé pour spécifier, visualiser, concevoir et documenter des systèmes d'information (SI) dans ses différentes dimensions et de manière standardisée. C'est plus qu'un outil, mais une norme qui s'est imposée dans le domaine de la modélisation objet(Hassas, 2017).

Pour mieux comprendre l'UML, il serait bien de faire un petit rappel historique sur celui-ci. Les méthodes utilisées dans les années 80 pour organiser la programmation fonctionnelle (notamment Merise) étaient fondées sur une modélisation séparée des données et des traitements(Dugerdil, 2005, p. 56). Lorsque la programmation objet prend de l'importance au début des années 90, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plusieurs méthodes apparaissent dans les années 90 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE etc.) mais aucune ne parviendra à s'imposer. En 1994, le consensus se fait autour de trois méthodes suivantes(Breu et al., 1997):

- OMT de James Rumbaugh fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
- OOD de Grady Booch, définie pour le Department of Defense, introduit le concept de package ;
- OOSE d'Ivar Jacobson fonde leur analyse sur la description des besoins des utilisateurs ;

C'est de cet effort de convergence qu'est né UML. Ce projet d'unification a évolué dans le temps. En 1995, Booch, Rumbaugh et d'autres chercheurs se sont mis d'accord pour construire une méthode unifiée nommée « Unified Method 0.8 » ; en 1996, Jacobson les a rejoints pour produire UML 0.9. Les grands acteurs de l'industrie du logiciel comme IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys s'associent alors à l'effort et UML 1.0 est soumis à l'Object Management Group (OMG), l'organisme international de normalisation en technologie objet. L'OMG adopte en 1997 UML 1.1 comme langage de modélisation des systèmes d'information (orienté objet). Et aujourd'hui, l'UML est à sa version 2.5.1(OMG, 2017).

Il offre une manière commune de représentation graphique d'un SI, facilitant ainsi une meilleure collaboration entre les concepteurs, les équipes de développement et les autres parties prenantes.

Divisés en deux groupes à savoir les diagrammes structurels et les diagrammes comportementaux qui symbolisent respectivement les vues statique et dynamique, les principaux diagrammes dont nous auront à utiliser sont les suivants :

- Le diagramme d'activité
- Le diagramme de cas d'utilisation
- Le diagramme de classe
- Et le diagramme de séquence

2.1.2. Le diagramme d'activité

Le diagramme d'activité est une représentation du ou des processus métiers qui sont encore appelés Business Process. C'est un ensemble d'illustrations qui guident le concepteur à travers les différentes étapes d'une tâche à effectuer par un acteur du système en question. Autrement dit, il met l'accent sur les activités internes du système, les relations qu'elles entretiennent et leurs impacts sur les objets.

En gros, le diagramme d'activité modélise les flux d'activités, les actions, les décisions et les chemins de contrôle dans un processus d'activité.

2.1.3. Le diagramme de cas d'utilisation

Le diagramme de cas d'utilisation met en évidence les interactions entre les acteurs que ce soient les utilisateurs ou systèmes externes et les cas d'utilisation qui par la suite seront les fonctionnalités du système informatique. Il montre comment les utilisateurs interagissent avec le système pour accomplir certaines tâches qui leurs sont affectées. Le diagramme de cas d'utilisation est comme une carte qui montre comment les différentes personnes ou systèmes interagissent avec système en question et quels types d'actions ils peuvent accomplir. Cela aide concepteurs et développeurs à avoir une vue d'ensemble claire des fonctionnalités.

2.1.4. Le diagramme de classe

Étant donné qu'un diagramme de cas d'utilisation représente un système du point de vue des acteurs, un diagramme de classe, lui décrit sa structure interne. C'est une méthode de

représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation. Il est considéré comme l'un des diagrammes les plus important de la modélisation orientée objet. C'est un diagramme de type structurel représenté par des classes et les relations entre ces classes. Il ne tient pas compte des comportements du système. Notons que tout système orienté objet est organisé autour des classes. Une classe est le formalisme d'un ensemble d'objets dotés de caractéristiques communes à savoir son état et son comportement, respectivement ses attributs et ses méthodes.

2.1.5. Le diagramme de séquence

Un diagramme de séquence permet de visualiser les interactions entre les acteurs et les objets d'un système. Il répond à la question suivante : qui fait quoi, dans quel ordre et comment ils communiquent entre eux ? L'aspect temporel y est primordial : c'est la vue dynamique. Les principales informations contenues dans ce diagramme sont les messages échangés entre les lignes de vie (acteurs, objets), présentés dans un ordre chronologique. Notons aussi que le diagramme de séquence matérialise des échanges entre objets dans le cadre d'un scénario d'un diagramme de cas d'utilisation.

2.2. Présentation du problème

2.2.1. Présentation d'un exemple de modèle de simulation d'une maladie infectieuse

Soit le modèle épidémiologique suivant à base d'Équations Différentielles Ordinaires (EDO) sur le Covid-19 au Sénégal de (Liu et al., 2020). Nous appelons ce modèle « Covid19sn ».

Soient S, Ai, Au, Ii et Iu les classes épidémiologiques représentant respectivement les populations des susceptibles, des infectés asymptomatiques répertoriés, des infectés asymptomatiques non répertoriés, des infectés symptomatiques répertoriés et des infectés symptomatiques non répertoriés, dans la gestion du Covid-19 au Sénégal. Le modèle mathématique à base d'équations et les valeurs de ses paramètres sont donnés ci-dessous.

$$\begin{cases} \dot{S}(t) = -\tau(t)S(t)[A_u(t) + I_u(t)] \\ \dot{A}_i(t) = p\tau(t)S(t)[A_u(t) + I_u(t)] - \upsilon A_i(t) \\ \dot{A}_u(t) = (1-p)\tau(t)S(t)[A_u(t) + I_u(t)] - \upsilon A_u(t) \\ \dot{I}_i(t) = \upsilon A_i(t) + f\upsilon A_u(t) - \eta I_i(t) \\ \dot{I}_u(t) = (1-f)\upsilon A_u(t) - \eta I_u(t) \end{cases}$$

$$S(t_0) = S_0 > 0 \; ; \; A_i(t_0) = A_{i0}; \; A_u(t_0) = A_{u0}; \; I_i(t_0) = I_{i0} \text{ and } I_u(t_0) = I_{u0}.$$

Équation 1. Modèle mathématique à base d'équations

Symbole	Interprétations	Méthode	Valeur
t_0	Début de l'épidémie	Estimé	-2.55
S_0	Population (nombre) des susceptibles à t0	Fixé	1.6 107
A_{i0}	Population (nombre) des infectés asymptomatiques répertoriés à t0	Estimé	5.9903
A_{u0}	Population (nombre) des infectés asymptomatiques non répertoriés à t0	Estimé	1.3299
I_{i0}	Population (nombre) des infectés symptomatiques répertoriés à t0	Estimé	0.1553
I_{u0}	Population (nombre) des infectés symptomatiques non répertoriés à t0	Estimé	0.0274
τ	Taux de transmission	Estimé	2.5994 10-8
$1/\upsilon$	Durée moyenne pendant laquelle les infections asymptomatiques sont asymptomatiques	Fixé	7
f	Fraction des infections asymptomatiques qui deviennent des infections symptomatiques signalées	Fixé	0.7
p	Fraction de personnes asymptomatiques signalées	Estimé	0.5
$1/\eta$	Durée moyenne d'apparition des symptômes d'une infection symptomatique	Fixé	10

Tableau 1. Définition des paramètres et des valeurs du modèle mathématique

2.2.2. Présentation d'un exemple de plateforme de simulation

Soit GAMA[3] (Drogoul et al., 2013), une plate-forme de simulation qui vise à fournir aux experts de terrain, aux modélisateurs et aux informaticiens un environnement complet de modélisation

[3] Le site web officiel de GAMA est : https://gama-platform.org/

et de développement de simulation pour créer des simulations multi-agents spatialement explicites.

GAMA propose:

- Un langage de modélisation complet, appelé GAML, pour la modélisation d'agents et d'environnements,
- Une grande bibliothèque extensible de primitives (mouvement de l'agent, communication, fonctions mathématiques, fonctionnalités graphiques, ...),
- Une reproductibilité multiplateforme des expériences et des simulations,
- Un puissant sous-système de dessin déclaratif et de traçage,
- Une interface utilisateur flexible basée sur la plateforme Eclipse,
- Un ensemble complet d'outils batch, permettant une exploration systématique ou "intelligente" des espaces paramètres des modèles.

Exécuter une *« expérience »* est le seul moyen, dans GAMA, d'exécuter des simulations sur un modèle. Dans GAMA, une « *expérience* » permet de définir la façon de simuler un modèle en fournissant les paramètres d'entrées du modèle pour lesquels l'utilisateur peut donner et modifier les valeurs et en spécifiant les sorties de simulations (comme les écrans, les moniteurs ou les inspecteurs). Les expériences peuvent être menées de différentes manières.

- La première, et la plus courante, consiste à lancer une expérience depuis la perspective Modélisation, en utilisant l'interface graphique utilisateur proposée par la perspective Simulation pour exécuter des simulations.
- La deuxième méthode permet de lancer automatiquement une expérience à l'ouverture de GAMA, en utilisant ensuite la même interface graphique utilisateur.
- La dernière méthode, connue sous le nom d'exécution sans-tête (headless), n'utilise pas l'interface graphique utilisateur et permet de manipuler GAMA entièrement à partir de la ligne de commande.

Les trois méthodes sont strictement équivalentes en termes de calculs (à l'exception de la dernière qui omet tous les calculs nécessaires au rendu des simulations sur les écrans ou dans l'interface graphique utilisateur). Ils diffèrent simplement par leur utilisation :

- Le premier est largement utilisé lors de la conception de modèles ou de la démonstration de plusieurs modèles.
- Le second est destiné à être utilisé lors de la démonstration ou de l'expérimentation d'un seul modèle.
- Le dernier est utile lors de l'exécution de grands ensembles de simulations, en particulier sur des réseaux ou des grilles d'ordinateurs ou quand on veut utiliser GAMA à partir d'un client quelconque.

C'est cette dernière façon d'utiliser GAMA qui nous intéresse dans le cadre de ce travail. Car elle permet d'exécuter la plateforme sans interface graphique en lui transmettant un modèle, les valeurs des paramètres de simulation et de récupérer ensuite les résultats.

2.2.3. Exemple d'une simulation du modèle « *Covid19sn* » dans GAMA headless

En mode headless, GAMA peut être considéré comme n'importe quelle commande *shell*, dont le comportement est contrôlé en lui passant 2 arguments : un fichier d'expérience et un répertoire de sortie.

a. Un fichier d'expérience

Un fichier d'expérience d'entrée, utilisé pour décrire le plan d'exécution du modèle à simuler, ses entrées et les sorties attendues.

Le mode headless utilise un fichier XML pour décrire le plan d'exécution d'un modèle. Ce fichier contient minimum 3 éléments essentiels :

- La ***simulation***. Cet élément du fichier d'expérience d'entrée permet de déterminer le modèle à simuler en indiquant son chemin d'accès (« ***sourcePath*** »), la condition d'arrêt des simulations en indiquant la durée, en cycle, de la simulation (« ***finalStep*** ») et l'expérience (« ***experiment*** ») à exécuter. Dans le fichier 1, il s'agit donc de simuler le modèle appelé « *Covid19sn* » se trouvant dans le répertoire « *myModels* » en exécutant l'expérience appelée « *Covid19sn* » avec une durée de 730 cycles (ce qui représente 2 ans, en considérant qu'un cycle représente une journée de simulation). L'« ***id*** » permet d'identifier la simulation. Il faut comprendre qu'il est possible de définir plusieurs simulations dans le même fichier d'expérience.

```
<?xml version="1.0" encoding="UTF-8"?>
<Experiment_plan>
    <Simulation id="2" sourcePath="./myModels /Covid19sn.gaml" finalStep="730" experiment="Covid19sn">
        <Parameters>
            <Parameter name="S0" type="INT" value="53" />
            <Parameter name="Ai" type="INT" value="621" />
            <Parameter name="Au" type="INT" value="621" />
            <Parameter name="Ii" type="INT" value="621" />
            <Parameter name="Iu" type="INT" value="621" />
            <Parameter name="beta" type="FLOAT" value="621" />
            <Parameter name="v" type="FLOAT" value="621" />
            <Parameter name="f" type="FLOAT" value="621" />
            <Parameter name="p" type="FLOAT" value="621" />
            <Parameter name="n" type="FLOAT" value="621" />
        </Parameters>
        <Outputs>
            <Output id="1" name="AiIi" framerate="1" />
            <Output id="2" name="AuIu" framerate="1" />
        </Outputs>
    </Simulation>
</Experiment_plan>
```

*Fichier 1. Plan d'exécution du modèle « **Covid19sn** » dans GAMA*

- Les **paramètres d'entrée**. Les paramètres d'entrée du fichier d'expérience décrivent les noms, types et valeurs des paramètres à transmettre au modèle pour son exécution. Dans le *fichier 1*, une valeur initiale est fixée pour chacun des paramètres (*Tableau 1*) du modèle « ***Covid19sn*** ».
- Les **sorties attendues**. Pour chaque sortie, il faut donner un identifiant, un nom, un type et la fréquence (exprimée en cycles) à laquelle elle est enregistrée dans le fichier de résultats pendant la simulation. Dans le *fichier 1*, nous enregistrons, tous les jours (chaque cycle : « *framerate="1"* »), le nombre de cas répertoriés qui est à la fois la population des symptomatiques répertoriés et des asymptomatiques répertoriés (*AiIi*) et le nombre de cas non répertoriés qui est à la fois la population des symptomatiques non répertoriés et des asymptomatiques non répertoriés (*AuIu*).

b. Un répertoire de sortie

Un répertoire de sortie, où seront stockés les résultats de la simulation.

2.2.4. Problématisation de l'automatisation du processus de simulation d'un modèle

Le modèle « ***Covid19sn*** » présenté dans la section 2.1 de ce chapitre est une représentation formelle, mathématiquement exprimée, qui n'a rien à voir avec une quelconque plateforme de simulation. A ce stade, le modèle est tout simplement décrit. Mais cette description est faite avec un langage mathématique. Ce qui pouvait se faire aussi avec un langage informatique ou tout simplement, dans la mesure du possible, avec un texte explicatif en français, en anglais, etc. Pour comprendre cette description, il faut donc comprendre le langage de description

utilisé. Mais cette description n'impose aucune contrainte particulière sur la structure du modèle que l'on peut vouloir simuler, pas plus qu'elle ne donne des détails particuliers sur le processus de simulation.

Mais la nécessité d'utiliser une plate-forme de simulation (dispositif informatique qui n'appartient ni au modèle ni au système de référence) capable d'interpréter le modèle et de produire les perturbations désirées sur ce modèle, va forcer à respecter certaines contraintes. Celles-ci sont de deux ordres (Treuil et al., 2008) :

- Des contraintes structurelles définies par le simulateur, qui vont imposer une certaine forme de description et de formalisation du modèle.
- Des contraintes d'accès qui spécifient ce qu'implique le fait pour le modèle de pouvoir être « perturbé » et qui vont également prescrire une certaine forme de description.

Les conséquences de ces contraintes font que la plateforme GAMA impose une certaine structure au fichier d'expérimentation d'entrée et aux types de données des paramètres d'entrée, ainsi qu'aux paramètres de sorties. Une autre plateforme de simulation pourrait proposer une structure différente pour le fichier d'expérimentation d'entrée et des types de données différents.

Par ailleurs, quand un utilisateur souhaite simuler un modèle à partir de notre application qui est un client web, il fournira les valeurs des paramètres d'entrée du modèle via un formulaire HTML. Or les types de données dans HTML ne correspondent pas forcément aux types de données des plateformes de simulation. A ce niveau, il faut donc, d'une part, pouvoir générer dynamiquement un formulaire HTML à partir des paramètres du modèle. D'autre part, il faut pouvoir faire une correspondance entre les types de données HTML et ceux des plateformes de simulation. Le même problème est posé quant à l'exploitation des résultats (sorties) de la simulation d'un modèle dans notre application.

2.3. Analyse du module « simulation manager »

2.3.1. Les fonctionnalités et les acteurs

Notre module de gestion des simulations est constitué de quatre groupes de fonctionnalités :

- Requérir une simulation :

- Acteur : le « gestionnaire des requêtes et sorties de simulations ». Il faut noter que ce dernier est le seul acteur qui interagit avec le « gestionnaire des simulations ». En effet, les utilisateurs de la plateforme, qu'ils soient des acteurs humains ou non (quand c'est par exemple un système de veille qui requiert des simulations), interagissent avec le « gestionnaire des simulations » par l'intermédiaire du « gestionnaire des requêtes et sorties de simulations ».
- Description : le « gestionnaire des requêtes et sorties de simulations » requiert une simulation en spécifiant le modèle, la plateforme de simulation, et les paramètres d'entrée. La simulation est ensuite initialisée et, selon le modèle à simuler, des données d'entrée de simulation peuvent être fournies pour que la simulation soit mise en œuvre.

- Fournir des données :

  - Acteur : le « gestionnaire des requêtes et sorties de simulations ».
  - Description : cette fonctionnalité permet de fournir des données d'entrées de simulation. Pour cela, un formulaire est dynamiquement créé et fourni au « gestionnaire des requêtes et sorties de simulation ». A partir de ce formulaire, un utilisateur peut saisir les données qui sont ensuite transmises au « gestionnaire des simulations » par le « gestionnaire des requêtes et sorties de simulations ».

- Exécuter la simulation :

  - Acteur : le « gestionnaire des requêtes et sorties de simulations ».
  - Description : cette fonctionnalité permet de mettre en œuvre une simulation. Il s'agit de l'exécution du modèle par la plateforme de simulation, avec les données fournies.

- Exploiter les résultats de simulation :

  - Acteur : le « gestionnaire des requêtes et sorties de simulations ».
  - Description : le « gestionnaire des requêtes et sorties de simulations » récupère les résultats de la simulation afin de permettre à l'utilisateur d'exploiter les résultats de sa simulation par le biais d'une visualisation via une interface utilisateur par exemple.

2.3.2. Le diagramme de cas d'utilisation

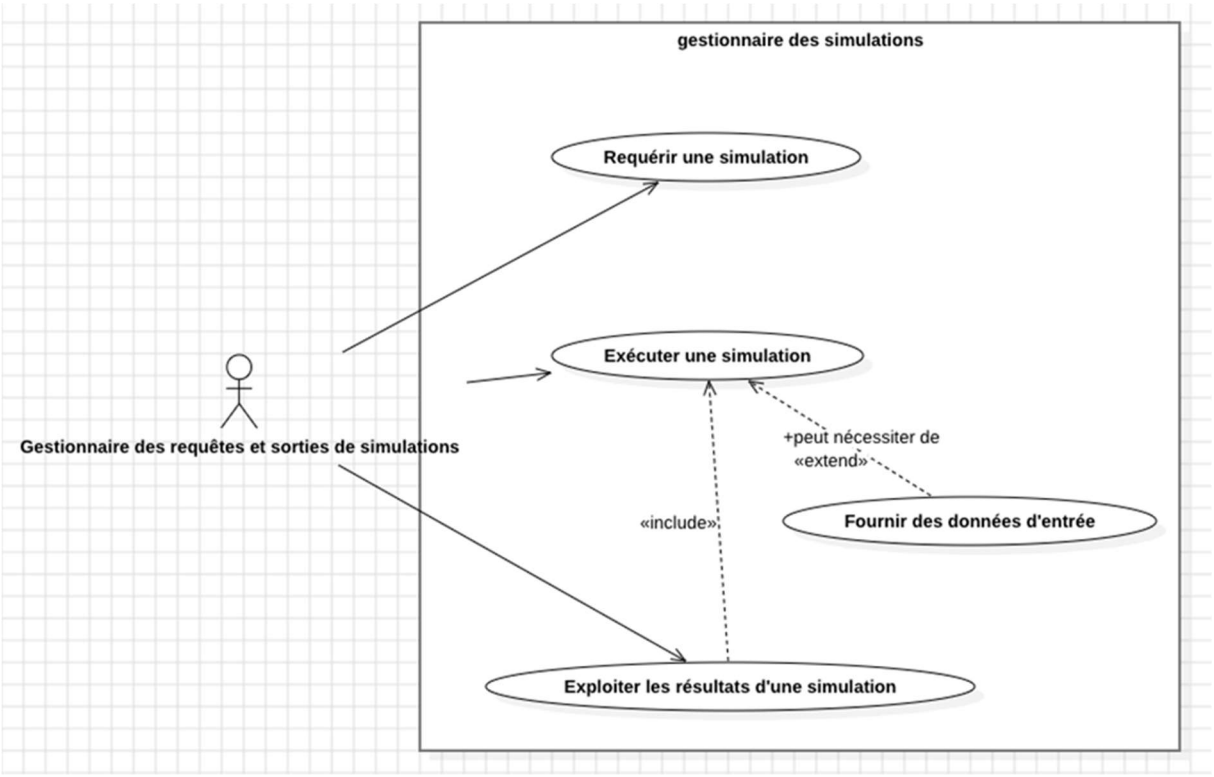

Figure 4. Diagramme de cas d'utilisation du gestionnaire des simulations "Simulation manager"

Dans notre module « gestionnaire des simulations », un seul acteur intervient dans le processus de simulation : le module « gestionnaire des requêtes et sorties de simulations ». Une fois que ce dernier requiert une simulation, il interroge d'abord le module en question en spécifiant le modèle, la plateforme de simulation et les paramètres d'entrée. Ensuite, un formulaire est dynamiquement créé (voire section 4.1.4.2 du chapitre 1), permettant de fournir les données d'entrée nécessaire. C'est là que l'exécution de la simulation est déclenchée avec les données fournies. Après simulation, le "gestionnaire des requêtes et sorties de simulations" récupère les résultats de simulation, permettant ainsi à l'utilisateur de les exploiter.

2.4. Conception du module « simulation manager »

2.4.1. Diagramme de classes

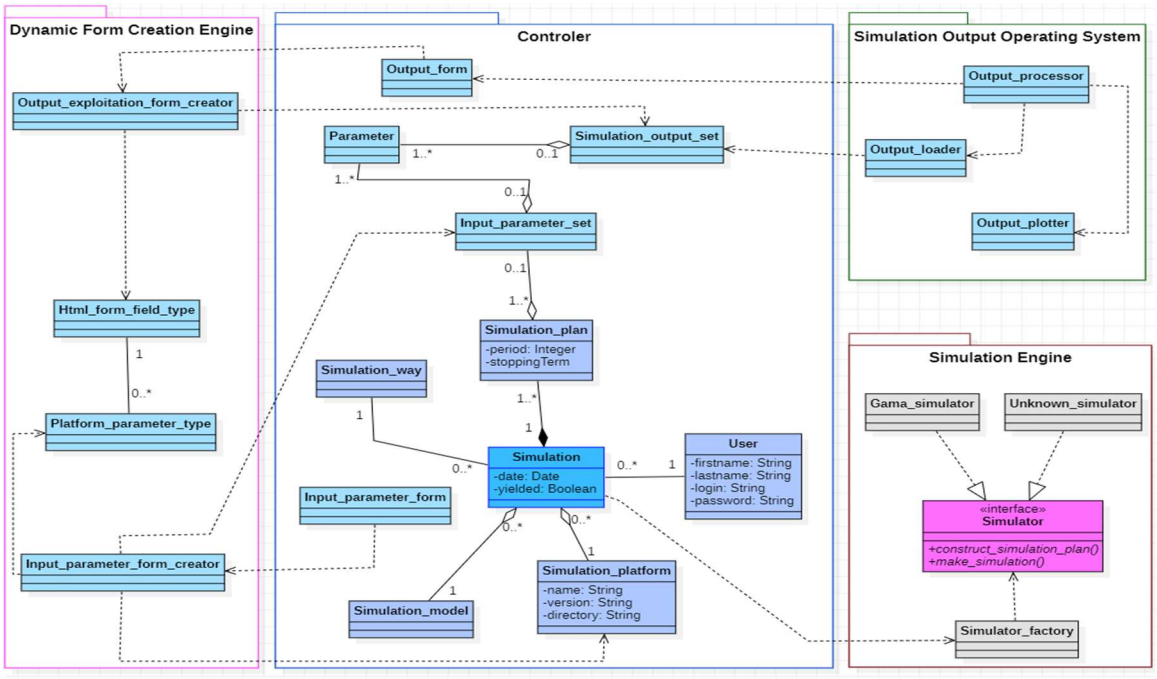

Figure 5. Diagramme de classes du gestionnaire des simulations "Simulation manager"

2.4.2. Description du diagramme de classes

Comme dans l'architecture de la plateforme présentée dans la section 4.1 du chapitre 1, les classes du gestionnaire des simulations sont réparties dans les 4 sous modules qui le composent : le moteur de simulation, le moteur de création dynamique de formulaires, le moteur d'exploitation des sorties de simulation et le contrôleur.

2.4.2.1. Le contrôleur

Le rôle du contrôleur est d'initialiser une simulation et d'orchestrer son déroulement. En effet, comme on peut le voir dans le diagramme de classes (voire section 4.1 de ce chapitre), avec les frontières des différents composants et les liens d'interactions (entrées/sorties) établis, les 3 moteurs sont indépendants et communiquent exclusivement avec le contrôleur. Le déroulement de l'orchestration d'une simulation est donné dans le diagramme d'activités (section 4.3 de ce chapitre). Au niveau du contrôleur, une simulation (classe "Simulation") est définie en fonction d'un modèle, d'une plateforme de simulation et d'un ensemble de plans de simulation.

- Un modèle à simuler (classe "Simulation Model") qui est donné comme entrée au gestionnaire des simulations par le « Sélecteur de modèles ». Dans ce diagramme, nous

avons choisi de ne pas donner des détails sur la description d'un modèle de simulation. En effet, cette description des modèles de simulation (on parlera aussi « d'annotation ») est assurée par l'ontologie et inclut celle des paramètres d'entrées (ceux pour qui il faut fournir des valeurs lors d'une simulation) du modèle ainsi que celle des résultats attendus (paramètres de sorties) dans une simulation.

- Une plateforme de simulation (classe "Simulation Platform") permettant de simuler le modèle. Elle est aussi donnée comme entrée au gestionnaire des simulations par le « Sélecteur de modèles ». Cette classe comporte entre autres les attributs suivants : le nom, la version, le répertoire d'accès, …
- Un ensemble de plans de simulation (classe "Simulation Plan"). Un plan de simulation comporte globalement les détails de la simulation d'un modèle dans une plateforme de simulation (durée, condition d'arrêt de la simulation, …). Chaque plan de simulation fera l'objet d'une unité de simulation. Quand une simulation comporte un seul plan de simulation, cela veut dire qu'il y'aura une seule "unité de simulation" (une des façons de d'effectuer une simulation : classe "Simulation Way"). Cependant, si elle en comporte plusieurs, cela veut dire qu'il y'aura une "simulation répétitive" (une autre des façons de simuler un modèle : classe "Simulation Way"). Un plan de simulation comporte également :
    - Un jeu de paramètres d'entrée (classe "Parameter Set") dont les valeurs sont fournies via un formulaire de saisie des valeurs de paramètres (classe "Parameter Form"). Ce formulaire de saisie est dynamiquement créé par le moteur de création dynamique de formulaires quand celui-ci est sollicité par le contrôleur à cet effet.
    - Un jeu de paramètres de sortie de simulation (classe "Simulation Output Set") dont les valeurs sont déterminées lors de l'exécution du « moteur de simulation ». Celui-ci est sollicité par le contrôleur en lui passant l'objet *simulation* qui comporte ce jeu de paramètres.
- L'utilisateur (classe "User") qui a effectué la simulation, la date de la simulation (attribut "date" et l'attribut "yielded" qui renseigne sur l'aboutissement de la simulation. Ces différents éléments permettront à un utilisateur de retrouver les simulations qu'il a effectué, de pouvoir retrouver et réexploiter les paramètres d'entrée et les résultats des simulations.

2.4.2.2. Le moteur de simulations

La conception du « moteur de simulations » se base sur le design pattern « Factory Method » qui sera présenté en détails dans la section 1.1.2. du chapitre 3. En effet, le moteur de simulation utilise des plateformes de simulations pour effectuer les *simulations*. Or comme nous l'avons vu dans la section 2.4 de ce chapitre, chaque plateforme de simulation détermine son propre plan de simulation. Cependant, on veut pouvoir utiliser des plateformes de simulation, sans avoir à appeler directement les constructeurs de leurs classes, de manière qu'on puisse modifier les constructeurs ou ajouter de nouvelles plateformes de simulation sans que le code qui utilise ces plateformes soit modifié. Pour cela, la classe "Simulator" est une *interface* dont implémente chaque plateforme de simulation ajoutée dans notre système (pour le moment, nous avons la classe "Gama Simulator" permettant de simuler avec la plateforme GAMA et la classe "Unknown Platform" permettant de gérer les *exceptions* liées au fait de vouloir simuler un modèle sans qu'une plateforme de simulation soit disponible). La classe "Simulator Factory" offre donc une méthode de création d'objets de type *Simulator*. C'est elle qui est invoquée par le contrôleur pour simuler un modèle.

2.4.2.3. Le moteur de création dynamique de formulaires

Le « moteur de création dynamique de formulaires » comporte:

- La classe "Parameter Input Form Creator" permettant de créer un formulaire de saisie (HTML) en se basant sur les paramètres d'entrée d'un modèle de simulation (classe "Simulation Model"), sur la plateforme de simulation (classe "Simulation Platform") à utiliser pour simuler le modèle et sur un mapping entre les types de données de la plateforme de simulation (classe "Platform Parameter Type") et les types de données HTML (classe "HTML Form Field Type").
- La classe "Output Exploitation Form Creator" permettant de créer un formulaire dont les champs correspondent aux résultats de sorties de simulation (classe "Simulation Output Set") et qui offre des *filtres* pour que l'utilisateur puisse exploiter ses résultats de simulation plus facilement.

2.4.2.4. Le moteur d'exploitation des résultats de simulation

Le « moteur d'exploitation des résultats de simulation » comporte les classes suivantes :

- "Output Loader" permettant de charger les sorties d'une simulation.

- "Output Processor" permettant de traiter les sorties chargées avec l'aide du formulaire d'exploitation des sorties de simulation.
- "Output Ploter" permettant, dans le traitement des résultats de simulation, de tracer des graphiques et des diagrammes.

2.4.3. Le diagramme d'activités

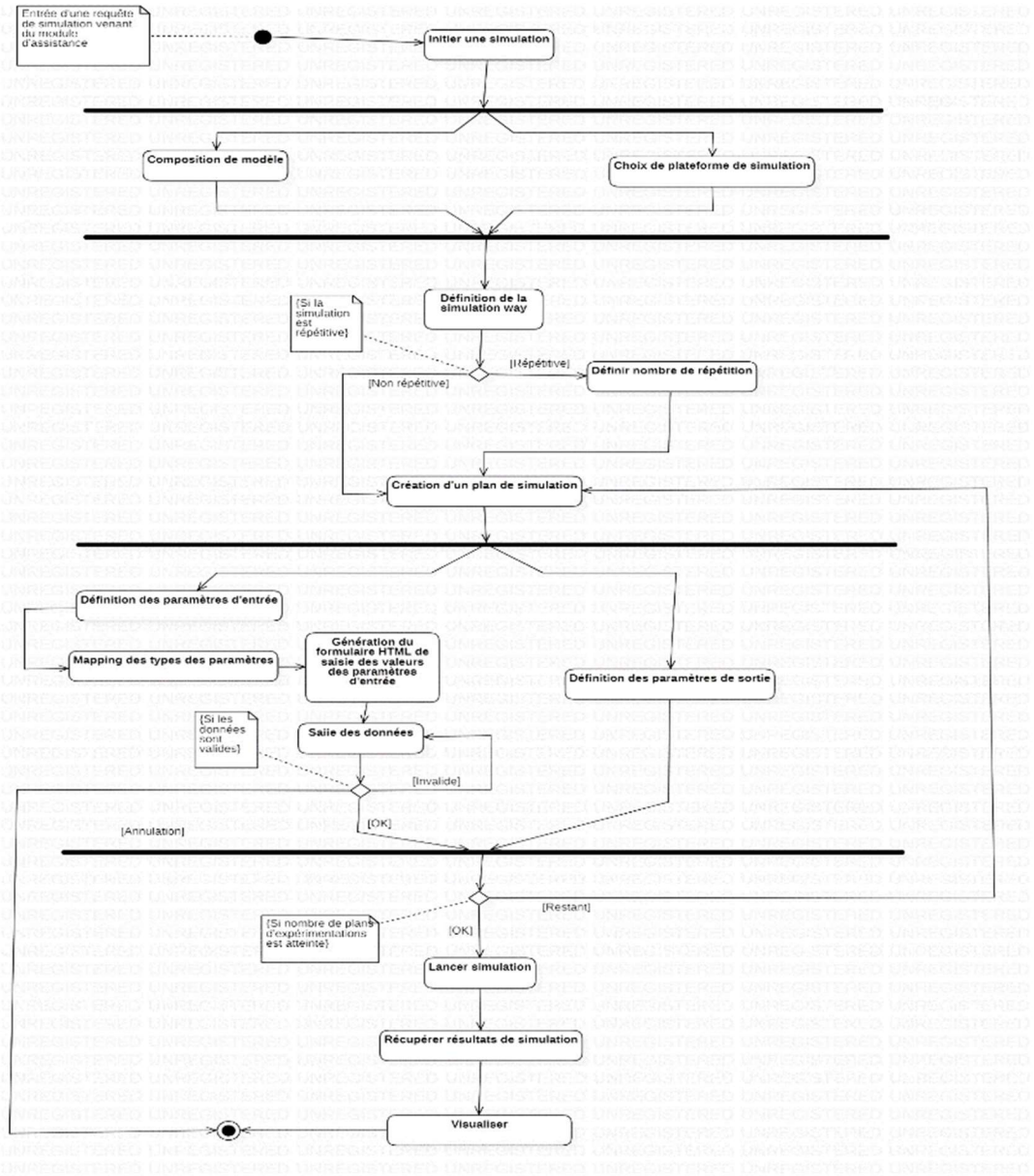

Figure 6. Diagramme d'activité du gestionnaire des simulations "Simulation manager"

Le processus de gestion des simulations dans le diagramme d'activités ci-dessus englobe toutes les étapes nécessaires pour initier et lancer une simulation, mais aussi pour récupérer et exploiter les résultats de cette simulation. Un seul acteur y est représenté: le « gestionnaire des

simulations »; c'est un peu comme le chef d'orchestre de la plateforme. C'est le point d'interaction principal entre les utilisateurs de notre plateforme de simulation et la plateforme elle-même. En requérant une simulation, il initie le processus. Les différentes activités du processus sont les suivantes:

- Initier une simulation : par l'intermédiaire du « gestionnaire des requêtes et sorties de simulations », une simulation est initiée en composant un modèle, en choisissant une plateforme de simulation et en définissant la façon de simuler le modèle « simulation way ».
- Création d'un ou des plans de simulation : le contrôleur crée un ou plusieurs plans de simulation en fonction de la « simulation way » initialement définie. De ce fait, il définit les paramètres d'entrée du modèle composé et les paramètres de sorties permettant l'exploitation des éventuels résultats de la simulation lancée.
- Génération dynamique du formulaire de saisie des paramètres d'entrée : le contrôleur génère dynamiquement un formulaire pour que l'utilisateur puisse saisir les valeurs des paramètres d'entrée.
- Saisie des données : l'utilisateur utilise l'interface qui lui est dédiée pour entrer les valeurs des paramètres d'entrée dans le formulaire.
- Lancer la simulation : le contrôleur transmet les données d'entrée et le modèle au moteur de simulation pour effectuer la simulation.
- Gérer les résultats de simulation : le contrôleur récupère les résultats de la simulation du moteur de simulation afin de permettre à l'utilisateur d'exploiter les résultats de sa simulation par le biais d'une visualisation qui est rendue possible grâce aux paramètres de sortie déjà définis.

Chapitre 3. Implémentation et présentation du « gestionnaire des simulations »

3.1. Outils et techniques de développements

3.1.1. Les Design Pattern

Un design pattern ou patron de conception est un modèle de solution réutilisable et éprouvée pour résoudre des problèmes de conception récurrents en développement logiciel. Ils sont catégorisés en plusieurs types en fonction de leur objectif et de leur application : il y'a des design pattern de création, de structure, de comportement et d'architecture.

3.1.1.1. Le Design Pattern MVC

MVC pour Model-Vue-Controller est un design pattern architectural largement utilisé dans le développement logiciel, particulièrement dans les applications orientées interface utilisateur. Il divise une application en trois composants principaux (le Modèle, la Vue, et le Contrôleur) pour organiser le code de manière modulaire.

- Le Modèle représente la structure des données de l'application, les règles métier et la logique d'accès et de manipulation des données.
- La Vue est responsable de la présentation des données au niveau de l'interface utilisateur. Elle affiche les informations au format approprié.
- Le Contrôleur traite les entrées de l'utilisateur, interagit avec le Modèle en conséquence, et met à jour la Vue en fonction des changements dans le Modèle. Il agit comme un intermédiaire entre la Vue et le Modèle.

Notons que l'objectif du design pattern MVC est principalement de rationaliser la communication entre les développeurs (Holovaty et al., 2008, p. 49).

3.1.1.2. Le Design Pattern "Factory method"

"Factory method" (Freeman and Freeman, 2010, p. 108:109) est un design pattern de création. Il permet de déléguer l'instanciation d'objets à des sous-classes, en fournissant une interface pour créer ces instances de classe. Mais le choix des classes concrètes à instancier est laissé aux sous-classes.

Nous avons une abstraction, définie par une interface commune (une classe abstraite sur python, Django), représentant différents types d'objets. Chacun de ces types d'objets doit être créé d'une manière spécifique qui peut différer d'un type à l'autre. Nous voulons éviter que le code client soit directement lié à ces implémentations concrètes et à leurs détails de création.

Le patron de conception "Factory Method" qui introduit une classe abstraite (le créateur) avec une méthode (le Factory Method) pour créer des objets conformes à une interface commune. Les classes concrètes dérivées de cette classe abstraite fournissent l'implémentation concrète du "Factory Method", permettant ainsi la création d'objets spécifiques.

Les principaux acteurs contenus dans le design factory sont:

- Le client. C'est la classe ou le module qui utilise la fabrique pour créer des objets. Il utilise l'interface du créateur pour créer des instances de produits sans avoir à spécifier explicitement la classe concrète du produit. Le client est ainsi isolé des détails de l'implémentation de la création d'objets.
- Le produit. C'est une classe abstraite qui définit l'interface des objets que la fabrique crée. Les sous-classes concrètes du produit implémentent cette interface pour produire des objets concrets.
- Le créateur. C'est une classe abstraite qui définit une interface pour la création d'objets, y compris le Factory Method lui-même. Ce Factory Method renvoie un objet de type produit. Les sous-classes concrètes du créateur implémentent ce Factory Method pour créer des instances spécifiques de produits. Chaque créateur peut produire un type spécifique de produit. Ainsi, le créateur fournit une interface commune pour la création d'objets, mais la responsabilité de la création réelle est déléguée aux sous-classes.

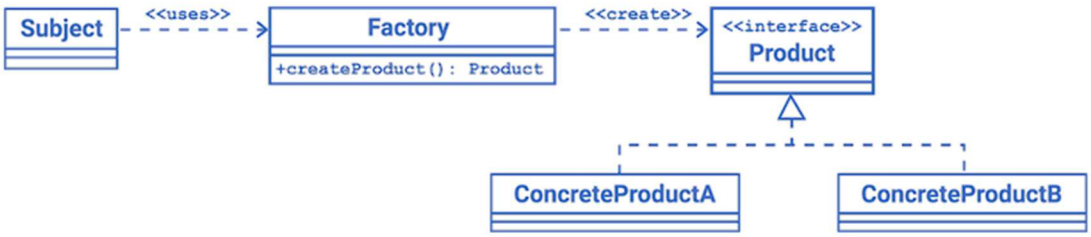

Figure 7. Design pattern Factory method

3.1.2. Django Framework

Le Framework Django (Holovaty et al., 2008) est un ensemble d'outils et de bibliothèques open-source conçu pour faciliter le développement rapide et efficace d'applications web dynamiques.

Il suit le paradigme architectural du modèle-vue-contrôleur (MVC) qui sépare les différentes responsabilités du développement web en trois composants distincts : le modèle pour représentation des données, la vue pour l'affichage des données et le contrôleur pour la logique de traitement.

Les modèles définissent la structure des données d'une application. Ils agissent comme des modèles pour les informations qu'une application va stocker, comme les utilisateurs, les articles de blog, les produits, etc.

Les vues définissent la manière dont les données sont présentées aux utilisateurs. Elles sont responsables de l'affichage des informations extraites des modèles.

Le contrôleur dans Django gère le flux de données entre les modèles et les vues, traite les demandes des utilisateurs et fournit les sorties, résultats appropriés.

Django est doté d'un système de gestion d'URL ou routage permettant de diriger les utilisateurs vers les vues appropriées en fonction des URLs qui sont saisis.

Il fournit aussi une interface d'administration facilitant la gestion des données de l'application. Et parlant de données, Django propose un ORM (Object Relational Mapping) intégré, qui facilite la manipulation des données de la base de données en utilisant des objets Python plutôt que des requêtes SQL. Sur le plan sécuritaire, la gestion des utilisateurs, l'authentification et les droits (rôles) sont minutieusement pris en charge.

Dans Django, la vue tient souvent le rôle du contrôleur.

En somme, Django est un Framework puissant qui facilite le monde du développement en fournissant une structure organisée, des outils préconçus et des fonctionnalités très satisfaisantes. Le workflow qui débute avec une requête (url) et qui se termine avec l'affichage d'une page html, se passe comme suit:

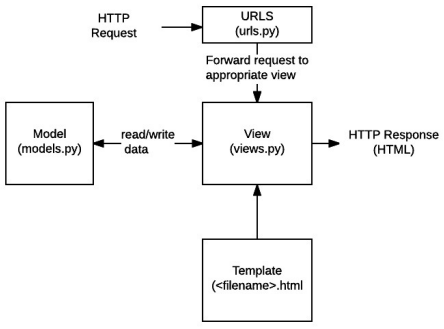

Figure 8. Workflow de Django Framework

- Le processus démarre lorsqu'un utilisateur envoie une requête HTTP à l'application Django. Cette requête est généralement initiée en tapant une URL dans le navigateur.
- La première étape consiste à faire correspondre l'URL de la requête à une vue spécifique. Cela est géré par le fichier " urls.py ". Django utilise un système de correspondance d'URL basé sur des expressions régulières pour diriger la requête vers la vue appropriée.
- Une fois que l'URL a été résolue, la vue correspondante est appelée. La vue contient la logique de l'application. Elle traite généralement la requête, interagit avec les modèles pour récupérer ou modifier des données, et renvoie une réponse.
- Les modèles définissent la structure de la base de données. Lorsqu'une vue nécessite des données, elle interagit avec les modèles pour récupérer ces données. Les modèles sont des classes Python qui représentent les tables de la base de données et définissent les champs et les relations entre les données.
- La vue traite la requête en fonction de la logique de l'application. Cela peut inclure la récupération de données depuis la base de données, le traitement des formulaires, ou d'autres opérations.
- Une fois que la vue a traité la requête, elle renvoie généralement une réponse HTML. Cette réponse est générée en utilisant un système de templates. Les templates sont des fichiers HTML avec des balises spéciales qui permettent d'incorporer dynamiquement des données générées par la vue.
- La réponse HTML générée est renvoyée au navigateur de l'utilisateur sous la forme d'une réponse HTTP.
- Enfin, le navigateur de l'utilisateur reçoit la réponse HTTP et affiche la page HTML générée.

3.1.3. Le Scaffolding

En développement web et plus précisément dans le cadre de l'utilisation des frameworks, le "scaffolding" se réfère à la génération automatique de la structure de base d'une application, y compris les fichiers, répertoires, modèles de code, et souvent la création des pages d'administration en prenant en charge les fastidieuses interfaces et opérations CRUD (Create, Read, Update, Delete) de base.

Cela réduit les efforts de développement en permettant aux développeurs de démarrer rapidement un nouveau projet en fournissant une base fonctionnelle sur laquelle ils peuvent construire toute leur application. Il est à noter que généralement ce sont des lignes de commande qui permettent de générer une telle structure.

En guise d'exemple, Django est la parfaite illustration. Dans celui-ci, le scaffolding se produit lorsque la commande "startproject" (Holovaty et al., 2008, p. 10) est exécutée pour créer un nouveau projet, suivie de la commande "startapp" (Holovaty et al., 2008, p. 52) pour créer une nouvelle application dans le projet en question.

En plus de la structure de base de l'application, le scaffolding peut être utilisé pour créer rapidement l'espace d'administration (application « admin » intégrée dans Django). L'espace d'administration de Django est une fonctionnalité intégrée qui fournit une interface d'administration générée automatiquement pour les modèles (models.py) de l'application. Cela permet aux administrateurs de gérer facilement les données sans avoir à créer manuellement des formulaires et des vues. Pour ce faire, il faut simplement activer l'application Django nommée "django.contrib.admin" en l'ajoutant dans la variable "INSTALLED_APPS" du fichier de configuration "settings.py", s'il n y est pas déjà. Ensuite, il faut ajouter son url dans le "contrôleur frontal" de votre projet qui est le fichier "urls.py" se trouvant dans notre projet Django.

Il s'en suit la création d'un "super-utilisateur" par le biais de la commande "python manage.py createsuperuser" où Django nous invitera à entrer les éléments d'identification de ce "super-utilisateur" : Nom d'utilisateur, adresse mail et mot de passe (à répéter). Enfin, pour accéder à notre interface d'administration, il suffit d'ajouter "*/admin/*" après le lien de notre projet. Cette fenêtre d'administration nous permettra de nous connecter avec les identifiants de notre "super-utilisateur" que nous venons de créer.

Pour administrer les différentes applications, il suffit d'ajouter les "models" de ces applications qui comportent des données de paramétrage sur lesquelles nous souhaitons effectuer des opérations CRUD. Pour cela, il faut signaler à Django pour quels "models" nous voulons disposer d'une interface d'administration. Nous allons le faire en créant un fichier "admin.py" dans les répertoires des applications.

3.1.4. PostgreSQL

PostgreSQL (Momjian, 2001) est un système de gestion de base de données relationnelle open-source et robuste. Il est conçu pour stocker et gérer de grandes quantités de données tout en offrant des fonctionnalités avancées de gestion, de requête et de traitement des données. Il offre des fonctionnalités avancées telles que la prise en charge des clés étrangères, des transactions répondant aux normes ACID (Atomicité, Cohérence, Isolation, Durabilité), des requêtes complexes et bien plus encore.

Cette technologie peut être étendu via des extensions et des langages de programmation, vous permettant d'ajouter des fonctionnalités personnalisées à la base de données et est compatible avec divers langages de programmation pour écrire des fonctions et des procédures stockées, y compris le langage SQL, Python et tant d'autres. Outre ses performances indiscutables et d'être open-source, PostgreSQL est réputé pour ses fonctionnalités avancées, ses performances élevées et son extensibilité.

3.1.5. Extensible Markup Language (XML)

XML (Marchal, 2002) est un format de données structuré utilisé pour stocker et échanger des informations entre différents systèmes informatiques. C'est un langage simple et lisible qui permet de décrire des données de manière organisée et hiérarchique. Une analogie pour mieux comprendre XML serait de le comparer à des étiquettes et des compartiments pour organiser des objets. XML vous permet de définir vos propres balises et de personnaliser la structure de vos données. Il est largement utilisé pour échanger des informations entre différents systèmes informatiques, étant donné qu'il fournit un format standardisé pour la transmission des données. En plus de cela, les applications informatiques peuvent facilement analyser et extraire des données à partir de documents XML, car les balises fournissent des indications claires sur la structure des données.

En gros, XML est comme un système d'étiquetage pour organiser des informations de manière hiérarchique. Les balises servent d'étiquettes pour identifier le contenu des données, et la structure hiérarchique permet de décrire la relation entre différentes parties des informations. C'est un format flexible et lisible par les humains, utilisé pour stocker et échanger des données entre les systèmes.

3.2. Implémentation du « gestionnaire des simulation »

Il faut noter que nos travaux de mémoire constituent la première étape dans la construction de la plateforme annoncée dans le chapitre 1. Pour cela, bien que nous nous focalisions sur le développement du module « gestionnaire des simulations », nous avons aussi initié le développement des autres modules qui doivent interagir avec celui-ci. Il s'agit de la base de données; du module « gestionnaire des requêtes et résultats de simulation » et des interfaces utilisateurs permettant de fournir des données d'entrées et d'exploiter les résultats de simulations.

Dans cette partie, nous présentons donc le développement de la plateforme avec le Framework Django et les détails d'implémentation du module « gestionnaire des simulations ».

3.2.1. Mise en place de la plateforme avec Django

Pour mettre en place une plateforme avec Django, il faut d'abord créer un « projet Django », puis y ajouter des « applications Django » qui constituent les modules applicatifs de la plateforme. Pour notre cas, le « projet Django » est créé et une application est ajoutée pour chacun des modules « gestionnaire des simulation » et « gestionnaire des requêtes et sorties de simulations ». Ensuite, avec Django, développer/implémenter une plateforme c'est développer/implémenter une-à-une chaque application ajoutée.

Par ailleurs, le scaffolding Django nous a fourni le module d'administration générale de la plateforme qui nous permet de gérer les données de paramétrage du module « gestionnaire des simulation ». C'est à partir de ce module qu'on gère (ajout, modification, suppression) les informations sur les plateformes de simulations pouvant être utilisées dans les simulations des modèles. Ces informations concernent, entre autres, les chemins d'accès des plateformes, les commandes de lancement, les types de données et leurs correspondances avec les types de données HTML.

3.2.2. Développement du module « gestionnaire des simulations »

Puisque Django se base sur l'architecture MVT (Model-View-Template, il s'agit de la version MVC de Django où le module "View" correspond au "Contrôleur" de MVC et le "Template" à sa "Vue"), développer une « application » c'est développer les "Model", les "View" et les "Template".

Dans notre plateforme, le module « gestionnaire des simulations » est une "application" Django. Pour la développer, il faut implémenter les composants "Model", "View" et "Template".

3.2.2.1. Développement du composant "Model" du « gestionnaire des simulation »

Dans Django, le développement du "Model" d'une application consiste à ajouter les différentes classes « métiers » de l'application dans le fichier "models.py". A priori, pour chaque classe ajoutée dans ce fichier, une table est créée par l'ORM Django dans la base de données. Ce qui permet d'enregistrer et de pérenniser un objet d'une classe donnée dans la table correspondante.

A partir du diagramme des classes du « gestionnaire des simulations » (Figure 5), les classes suivantes sont implémentées dans le fichier « models.py »:

```
class SimulationtPlan(models.Model):
class SimulationOutput(models.Model):
class SimulationWay(models.Model):
class Simulation(models.Model):
class SimulationPlatform(models.Model):
class HtmlFormFieldType(models.Model):
class PlatformParameterType(models.Model):
```

Figure 9. Les différentes classes métier du "gestionnaire des simulations"

- "Simulation", pour pouvoir enregistrer et garder en base de données toutes les simulations effectuées dans la plateforme.
- "SimulationPlan", pour pouvoir enregistrer et garder en base de données tous les plans de simulations élaborés dans la plateforme. Ce qui permettra à un utilisateur de retrouver l'historique de ses plans de simulations et d'initialiser de nouvelles simulations sur leurs bases.
- "SimulationOutput", pour pouvoir enregistrer et garder en base de données tous les résultats des simulations effectuées dans la plateforme. Ce qui permettra à un utilisateur de retrouver ses résultats de simulation sans avoir besoin de les reprendre.
- "SimulationWay", pour pouvoir gérer (ajouter, modifier, supprimer), depuis le module d'administration, les façons de simuler un modèle.
- "SimulationPlatform", pour pouvoir gérer (ajouter, modifier, supprimer), depuis le module d'administration, les plateformes de simulation pouvant être utilisées pour simuler les modèles.

- "PlatformParameterType", pour pouvoir gérer (ajouter, modifier, supprimer), depuis le module d'administration, les types de données des plateformes de simulation.
- "HTMLFormFieldType", pour pouvoir gérer (ajouter, modifier, supprimer), depuis le module d'administration, les correspondances entre les types de données des plateformes de simulation et les types de données HTML.

3.2.2.2. Développement des "View" du « gestionnaire des simulations »

Une vue ("View") est une fonction définie dans un fichier « views.py » et qui reçoit comme paramètres la requête (l'URL) à partir de laquelle elle est invoquée et d'éventuels autres paramètres à utiliser dans les traitements de la fonction. Les traitements effectués par la vue sont donnés dans le corps de la fonction. La vue rend enfin un template (une page HTML) qui correspond à l'url de départ ou une redirection vers une autre vue.

Concernant le « gestionnaire des simulations », nous avons les trois fonctionnalités suivantes qui correspondent à trois "URLs" dans le fichier "urls.py" et donc à trois "View" dans le fichier "views.py" de l'application:

```
urlpatterns = [
    path('simulationunit/<int:idModel>/<int:idPlatform>/', views.simulationunit, name='simulationunit'),
    path('repetitivesimulation/<int:idModel>/<int:idPlatform>/', views.repetitivesimulation, name='repetitivesimulation'),
    path('simulationoutputs/<int:idSimulation>/', views.simulationoutputs),
]
```

Figure 10. Urls correspondant aux fonctionnalités du "gestionnaire des simulations" dans son fichier « urls.py »

- Une unité de simulation ("simulationunit"). Il s'agit ici de la requête (URL) permettant d'effectuer une unité de simulation (une des « façons de simuler un modèle) d'un modèle ("idModel") avec une plateforme ("idPlatform").
- Une simulation répetitive ("repetitivesimulation"). Il s'agit ici de la requête (URL) permettant d'effectuer une simulation répétée (une autre des « façons de simuler un modèle) d'un modèle ("idModel") avec une plateforme ("idPlatform").
- Les résultats d'une simulation ("simulationOutputs"). Il s'agit ici de la requête (URL) permettant d'exploiter les résultats d'une simulation ("idSimulation").

Les vues ("view") correspondant à ces URLs sont les suivantes:

```
def simulationunit(request, idModel, idPlatform):...

def repetitivesimulation(request, idModel, idPlatform):...

def simulationoutputs(request, idSimulation):...
```

Figure 11. Les vues du "gestionnaire des simulations" dans son fichier "views.py"

- La vue permettant d'effectuer une unité de simulation ("simulationunit"). Cette fonction reçoit le modèle à simuler et la plateforme de simulation et initialise une "simulation" (instanciation d'un objet de la classe "Simulation" et son ajout en base de données). Elle fait appel ensuite au « Moteur de création dynamique de formulaire » (les détails de ce moteur sont donnés dans la section 2.2.2.1) en lui transmettant le modèle et la plateforme de simulation afin de constituer un formulaire permettant de saisir les valeurs des paramètres d'entrée du modèle à simuler. Ce formulaire est ensuite transmis au "template de saisie de paramètres d'entrée" (voire le rendu à la section 3.1). Avec ces paramètres, le « moteur de simulation » (les détails de ce moteur sont donnés dans la section 2.2.2.2.) est invoqué pour d'abord constituer un "Plan de simulation" pour la "simulation" et ensuite procéder à la mise en œuvre de la "simulation" pour ensuite ajouter les résultats dans l'objet "simulation". Une fois la simulation terminée, l'objet "simulation" est transmis à la vue permettant d'exploiter les résultats de simulation.
- La vue permettant d'effectuer une simulation répétitive ("repetitivesimulation"). Elle fonctionne de la même manière que la précédente, à peu de détails techniques prêts qui sont relatives à la "parallélisation des simulations" que nous ne détaillons pas ici. Cette partie sera plutard illustrée à la section 3.2 de ce chapitre.
- La vue permettant d'exploiter les résultats des simulations ("simulationoutputs"). Cette fonction reçoit une "simulation", fait appel au « Moteur de création dynamique de formulaire » afin de constituer un formulaire, conformément aux paramètres de sorties des simulations, permettant d'avoir un dispositif de filtre sur les résultats (bruts) des simulations. Ce formulaire est ensuite transmis au "template d'exploitation des résultats de simulation" (voire les rendus à la section 3.3). Les traitements à faire sur les résultats des simulations sont gérés par le « moteur d'exploitation des résultats de simulations » (les détails de ce moteur sont donnés dans la section 2.2.2.3.)

3.2.2.2.1. Le « moteur de création dynamique de formulaires »

Le « moteur de création dynamique de formulaires[4] » permet de constituer dynamiquement des formulaires HTML, conformément aux paramètres (d'entrées et de sorties) d'un modèle, en

[4] Il s'agit tout simplement d'un fichier Python « .py » comportant un ensemble de classes applicatives. C'est aussi le cas du « moteur de simulation » et du « moteur d'exploitation des résultats de simulations ».

s'appuyant sur les types de données des plateformes de simulation et sur leurs correspondances avec les types de données des champs (balises d'entrée) HTML.

```
class DynamicFormCreatorOnUnitSimulation(forms.Form):
    def __init__(self, model_to_simulate, simulation_platform, *args, **kwargs):

class DynamicFormCreatorOnRepetitiveSimulation(forms.Form):
    def __init__(self, model_to_simulate, simulation_platform, *args, **kwargs):

class OutputPlotFiltersForm(forms.Form):
    def __init__(self, mapped_output_names_and_labels, mapped_output_names_and_features, *args, **kwargs):
```

Figure 12. Les différentes classes du "moteur de création dynamique de formulaires"

Comme le montre la figure ci-dessus, ce moteur comporte:

- Une classe ("DynamicFormCreatorOnUnitSimulation") pour constituer un formulaire HTML conformément aux paramètres d'entrée d'un modèle ("model_to_simulate") et des types de données de la platefrome de simulation ("simulation_platform").
- Une classe ("DynamicFormCreatorOnRepetitiveSimulation") pour constituer un formulaire HTML conformément aux paramètres d'entrée d'un modèle ("model_to_simulate") et des types de données de la platefrome de simulation ("simulation_platform"), dans le cadre d'une simulation répétée.
- Une classe ("OutputPlotFiltersForm") pour constituer un formulaire HTML conformément aux paramètres de sorties d'un modèle. Cette classe s'appuie sur l'ensemble des paramètres de sortie ("mapped_output_names_and_labels") et sur leurs caractéristiques ("mapped_output_names_and_futures").

3.2.2.2.2. Le « moteur de simulation »

Nous avons implémenté le « moteur de simulation » en utilisant le design pattern « Factory method ». En effet, le moteur de simulation s'appuie sur un ensemble de plateformes de simulations qui déterminent un plan de simulation spécifique. Or, nous avons besoin de pouvoir ajouter de nouvelles plateformes et aussi modifier les codes des classes qui les représentent sans que le code qui utilise ces plateformes (ce code, appelé "Subject" se trouve dans les vues « view ») soit modifié.

```
class ModelSimulator(ABC):
    def __init__(self, model_to_simulate, simulation_platform, input_parameters, simulation):

    @abstractmethod
    def construct_experiment_plan(self):
        pass

    def make_simulation(self):

class GamaSimulator(ModelSimulator):

    def __init__(self, model_to_simulate, simulation_platform, input_parameters, simulation):
        super().__init__(model_to_simulate, simulation_platform, input_parameters, simulation)

    def construct_experiment_plan(self):

class UnknownPlatformSimulator(ModelSimulator):
    def __init__(self, model_to_simulate, simulation_platform, input_parameters, simulation):


    def construct_experiment_plan(self):

    def make_simulation(self):

class Experiment():
    def __init__(self, model_to_simulate, simulation_platform, input_parameters, simulation):
        if (simulation_platform == None):
            self.platform_simulator = UnknownPlatformSimulator(model_to_simulate, simulation_platform, input_parameters, simulation)
        elif (simulation_platform.platform_name == 'GAMA'):
            self.platform_simulator = GamaSimulator(model_to_simulate, simulation_platform, input_parameters, simulation)
        else:
            self.platform_simulator = UnknownPlatformSimulator(model_to_simulate, simulation_platform, input_parameters, simulation)
```

Figure 13. Les différentes classes du "moteur de simulations"

Pour cela, la classe "ModelSimulator" de la figure ci-dessus est ce qui est appelé dans le design pattern « Fatcory method » le "Product" : une classe abstraite (pour nous une "*interface*") dont implémente chaque plateforme de simulation ajoutée dans notre système.

Les classes "GamaSimulator" et "UnknownPlatformSimulator" sont les seules "ConcreteProduct", c'est-à-dire, les simulateurs concrets utilisés en ce moment. On peut voir qu'elles implémentent chacune leurs propres manières de construction de plan de simulation.

Enfin, la classe "Experiment" est le "Factory" qui offre les services de création de simulateurs (objets de types "ModelSimulator"). C'est cette classe qui est utilisé par le "Subject".

3.2.2.2.3. Le « moteur d'exploitation des résultats de simulations »

Le « moteur d'exploitation des résultats de simulations » permet à un utilisateur d'exploiter, via une interface, les sorties de ses simulations.

```
class SimulationOutputLoader():
    def __init__(self, simulation, simulated_model):

class SimulationOutputPloter():
    def __init__(self, data):
```

Figure 14. Les différentes classes du "moteur d'exploitation des résultats de simulations"

Pour le moment, comme le montre la figure ci-dessus, les seuls services de traitements qu'offre ce module sont:

- Le chargement des données brutes de sorties de simulations (la classe "SimulationOutputLoader").
- L'affichage de graphiques (courbes, diagrammes, etc.) sur les sorties de simulations (la classe "SimulationOutputPloter").

3.2.2.3. Développement des "Templates" du « gestionnaire des simulations »

Les trois « templates » associées aux trois vues du « gestionnaire des simulations » sont des pages « HTML » dont nous choisissons de présenter les « rendus » au lieu de se pencher sur les « codes ». Il s'agit des pages de "saisie des paramètres d'entrée d'un modèle dans une unité de simulation" (section 3.1), de "définition des grilles de valeurs des paramètres d'un modèle dans une simulation répétitive " (section 3.2) et de "exploitation des résultats de simulations" (section 3.3).

3.3. Présentation du « gestionnaire des simulations »

Nous présentons ci-dessous le « gestionnaire des simulations » en montrant l'interface de saisie des paramètres d'entrée d'un modèle dans une unité de simulation; l'interface de définition des intervalles des paramètres d'un modèle dans une simulation répétitive et l'interface d'exploitation des résultats de simulations. Pour cela, nous utilisons le modèle de simulation « Covide-19 sn » et la plateforme GAMA présentés respectivement dans les sections 2.1 et 2.2 du chapitre 2.

3.3.1. La page de « saisie des paramètres d'entrée d'un modèle » dans une unité de simulation

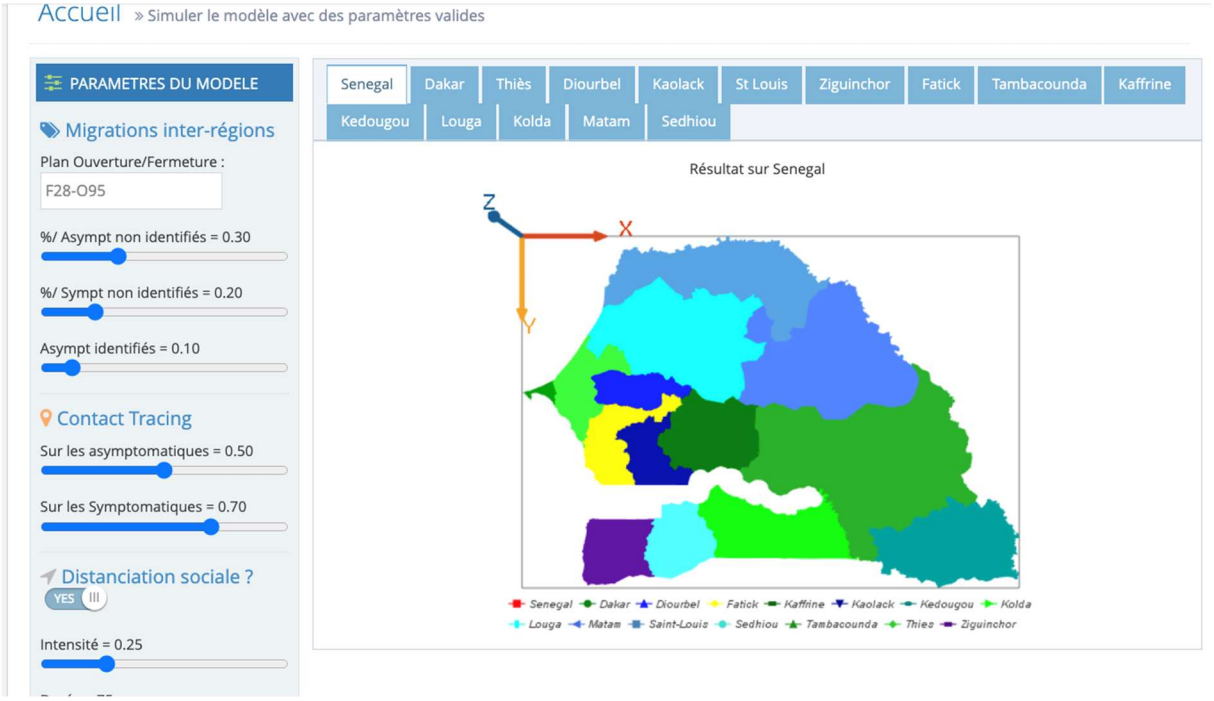

Figure 15. Page de saisie des valeurs des paramètres d'entrée d'un modèle dans une unité de simulation

Cette page montre à gauche le formulaire de saisie des valeurs des paramètres d'entrée du modèle créé dynamiquement par le « moteur de création dynamique de formulaires » à partir du modèle de la section 2.1 du chapitre 2 et la plateforme GAMA. Quand un utilisateur fournit ses valeurs d'entrée, il clique sur un bouton (non visible ici, car le reste du formulaire n'est pas capturé) qui lance les simulations : ce que l'on peut voir dans la figure suivante où la plateforme GAMA s'exécute en arrière-plan.

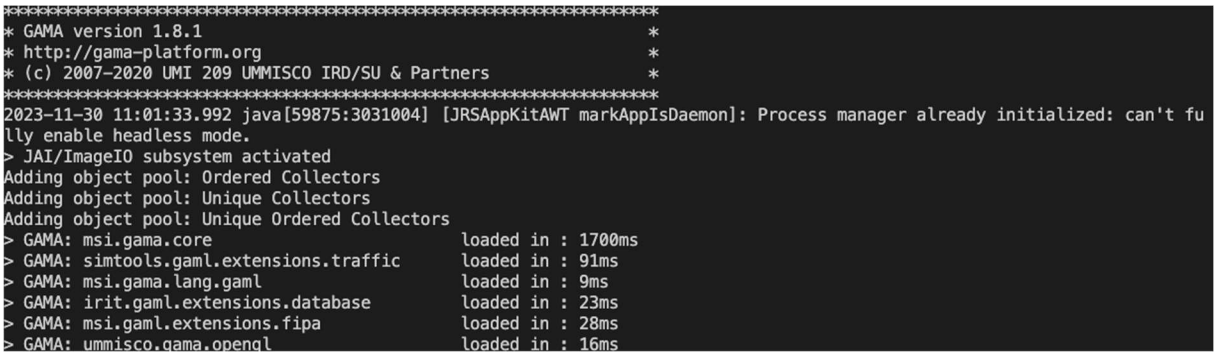

Figure 16. Exécution de la plateforme de simulation GAMA en arrière-plan

3.3.2. La page de définition des grilles de valeurs des paramètres d'un modèle dans une simulation répétitive

Comme on peut le voir dans les figures suivantes, cette page s'exécute en quatre itérations. Il faut d'abord choisir les plages de valeurs des paramètres pour lesquels une grille de valeurs doit être définie pour les simulations répétitives et définir le nombre de simulation à effectuer.

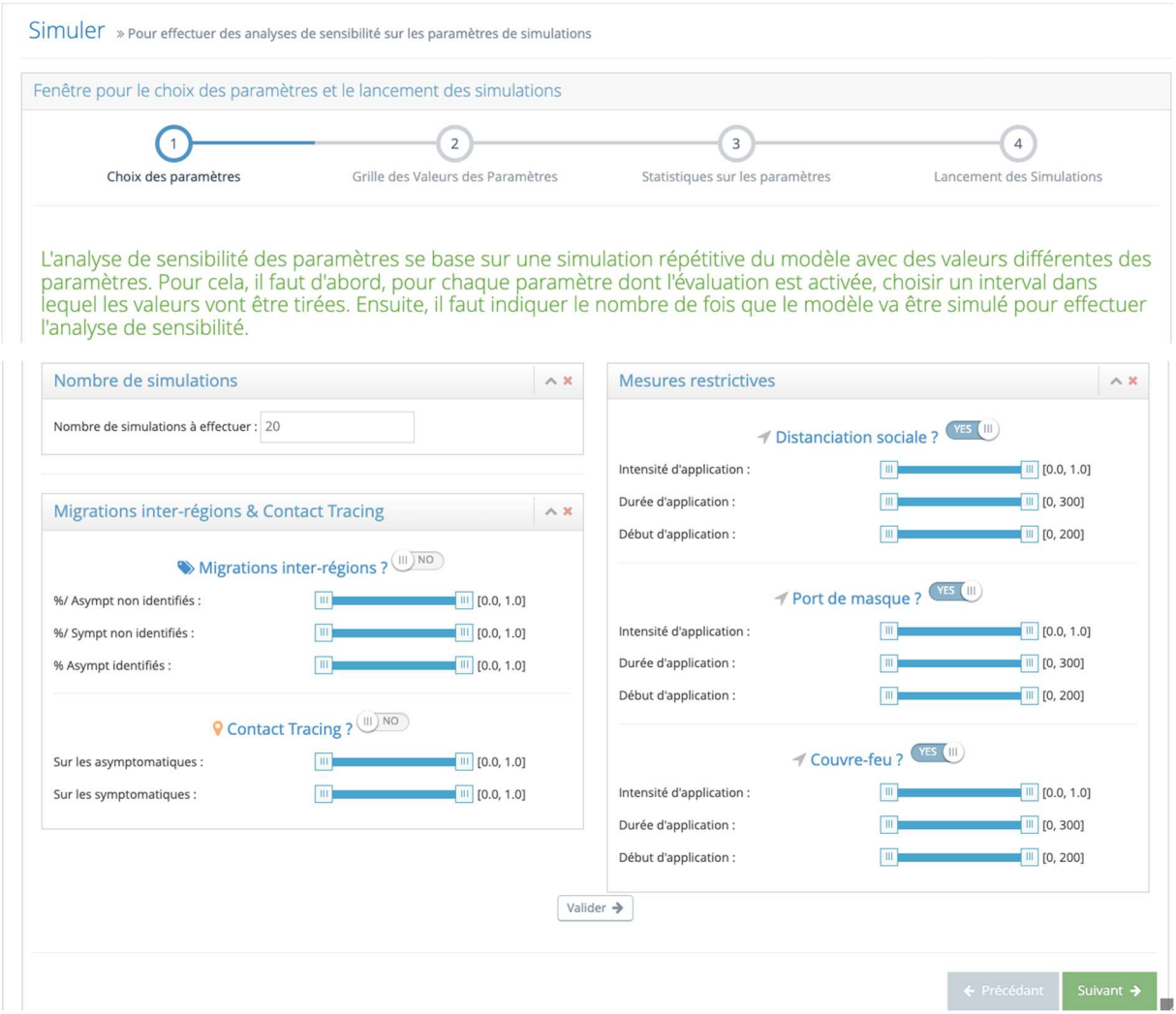

Figure 17. Page de choix des plages de valeurs des paramètres d'entrée d'un modèle dans une simulation répétitive

En cliquant sur le bouton « Valider », la page qui suit indique que les choix de l'utilisateurs sont bien pris en compte et l'invite à passer à l'étape suivante.

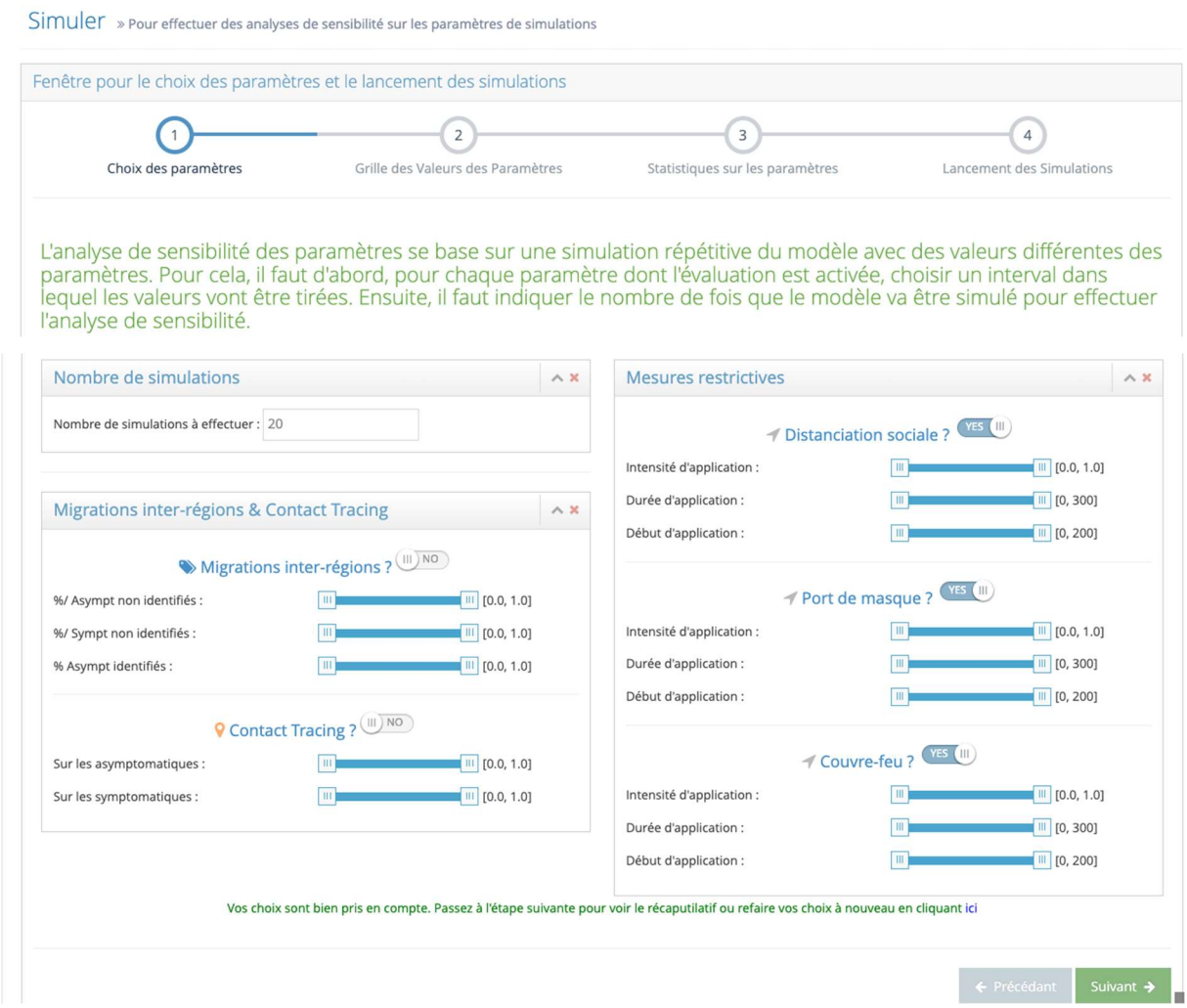

Figure 18. Page de confirmation des plages de valeurs des paramètres d'entrée d'un modèle dans une simulation répétitive

La page qui suit donne le tableau récapitulatif des grilles de valeurs des différents paramètres d'entrée du modèle choisis par l'utilisateur.

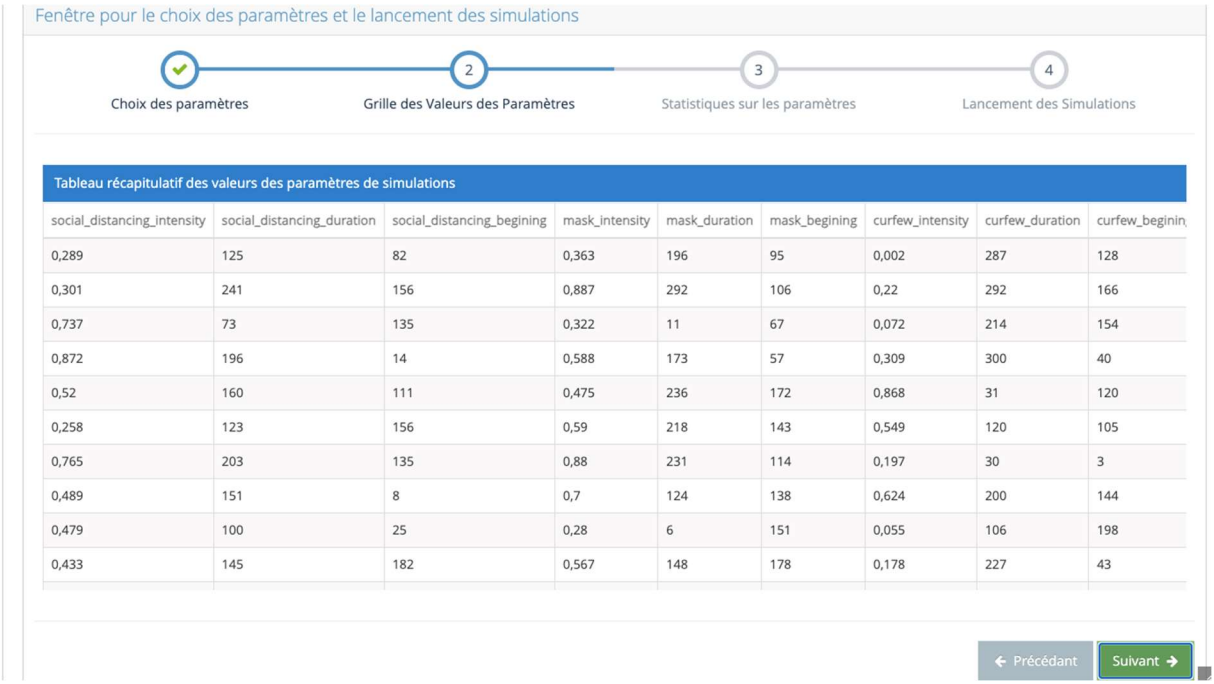

Figure 19. Page de la grille de valeurs des paramètres d'entrée d'un modèle dans une simulation répétitive

En cliquant sur le bouton, « Suivant », la page qui suit permet à l'utilisateur d'avoir des statistiques sur cette grille de valeurs afin de juger de la pertinence des données d'entrée qui lui sont proposées.

Figure 20. Page de présentation de statistiques sur la grille de valeurs des paramètres d'entrée d'une modèle dans une simulation répétitive

Cliquer sur le bouton « Suivant » de cette page permet de lancer la simulation répétitive.

3.3.3. La page d'exploitation des résultats de simulations

Comme nous l'avons expliqué dans la section 2.2.2.3 de ce chapitre, le seul traitement que le « moteur d'exploitation des résultats de simulations » propose en ce moment à l'utilisateur est l'affichage des données de sortie sous formes de graphiques avec des filtres.

Dans l'exemple ci-dessous, on peut voir les courbes d'évolution des différents compartiments épidémiologiques (du modèle « Covid-19sn ») avec la possibilité de les filtrer suivant les différentes régions du Sénégal.

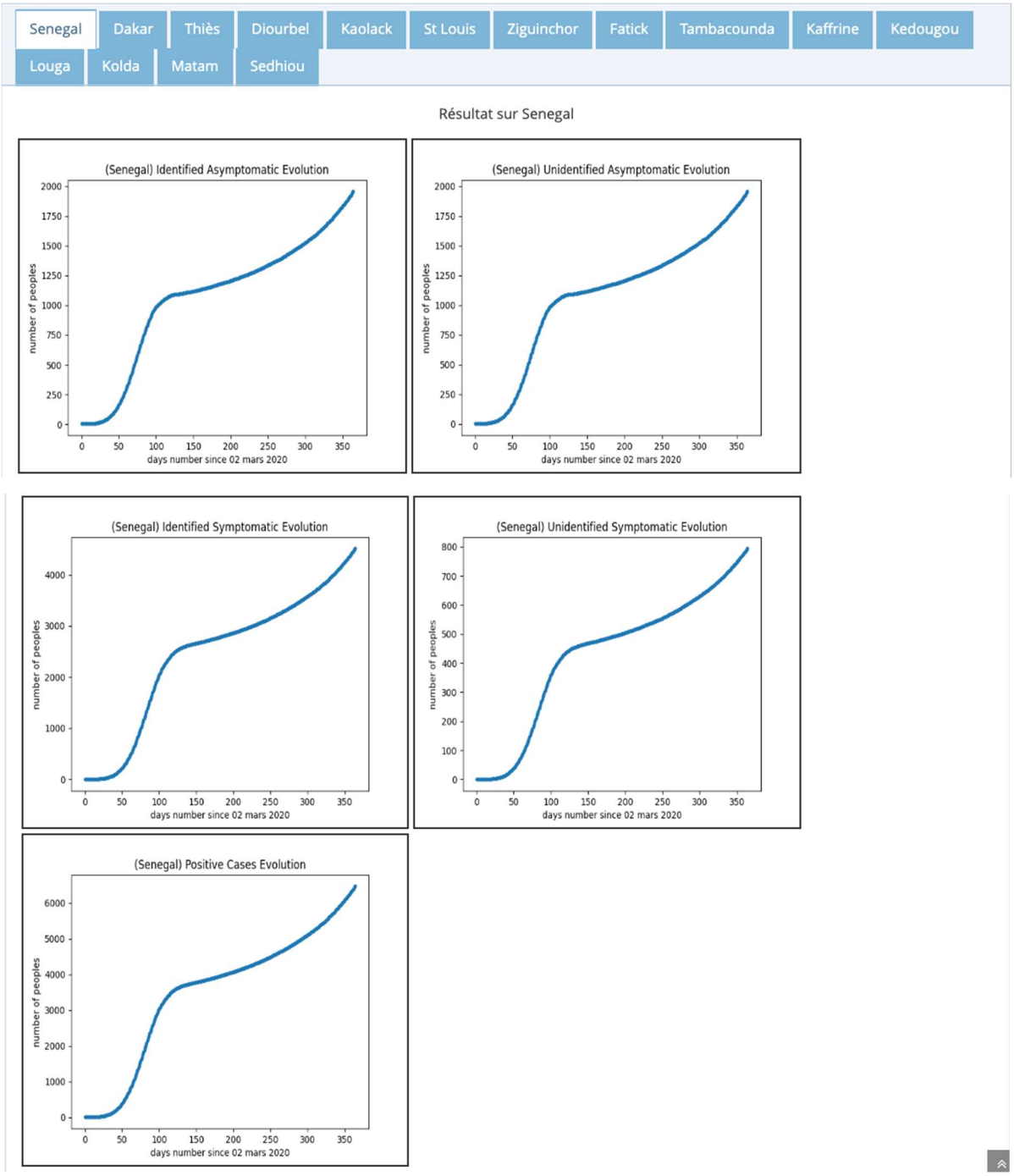

Figure 21. Résultats de simulation du modèle "Covid-19sn" montrant les courbes d'évolution des différents compartiments épidémiologiques dans le cadre du Sénégal

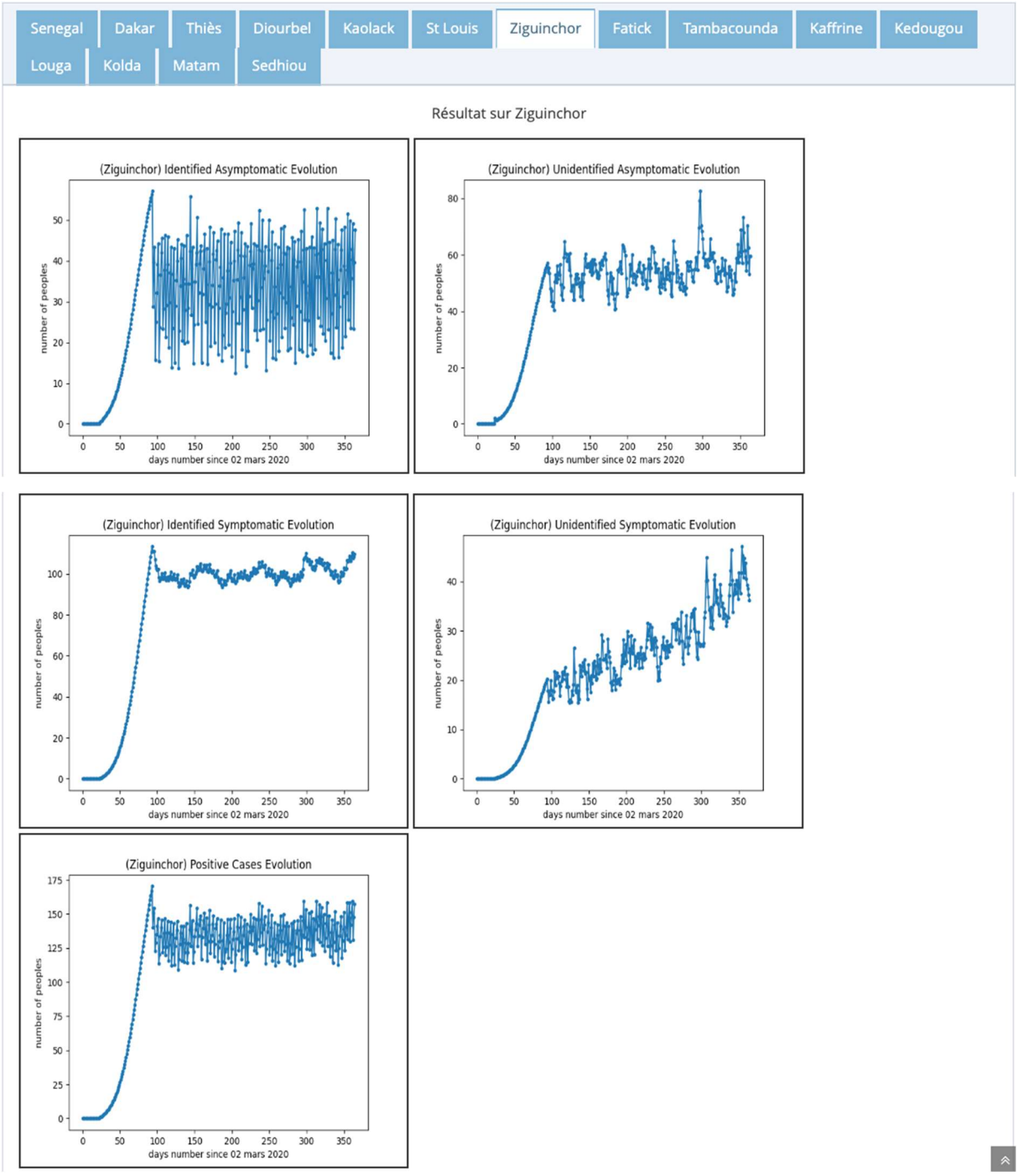

Figure 22. Résultats de simulation du modèle "Covid-19sn" montrant les courbes d'évolution des différents compartiments épidémiologiques dans le cadre de la région de Ziguinchor

CONCLUSION ET PERSPECTIVES

Ce projet vise à créer une plateforme novatrice basée sur des ontologies pour automatiser le processus de simulation des modèles de maladies infectieuses. Les objectifs spécifiques comprennent la mise en place d'une bibliothèque de modèles, la construction d'une ontologie pour annoter ces modèles, et enfin, l'implémentation d'un module d'orchestration et d'automatisation des simulations. Notre contribution principale a été le développement du module de gestion des simulations, ainsi que l'initialisation des modules associés tels que le gestionnaire des requêtes et sorties de simulations, les interfaces utilisateur, et la base de données.

Le module de gestion des simulations offre des fonctionnalités clés telles que la requête de simulations, la fourniture de données d'entrée, l'exécution des simulations, et l'exploitation des résultats obtenus. Cela permet aux utilisateurs de la plateforme, qu'ils soient humains ou systèmes automatisés, d'interagir de manière efficace avec les modèles de maladies infectieuses, facilitant ainsi le processus de simulation. Au-delà de la réalisation technique, notre travail contribue à la réalisation des objectifs généraux en favorisant la collaboration, la réutilisation des modèles, et en offrant un support de recherche. Ces avancées ouvrent la voie à une meilleure compréhension et modélisation des maladies infectieuses, tout en offrant un outil robuste pour la simulation et la recherche collaborative dans ce domaine.

Comme tout travail scientifique, notre mémoire comporte des limites. La modélisation des maladies infectieuses reste un domaine en constante évolution, et notre plateforme peut nécessiter des ajustements pour prendre en compte de nouveaux modèles ou des avancées technologiques. De plus, la qualité des résultats de simulation dépend de la précision des modèles eux-mêmes, soulignant l'importance continue de la recherche en épidémiologie pour affiner nos approches.

En regardant vers l'avenir, nous entrevoyons plusieurs perspectives passionnantes. Tout d'abord, l'expansion de notre bibliothèque de modèles pour inclure une diversité encore plus grande de maladies et de scénarios épidémiologiques. Ensuite, l'intégration de l'ontologie pour enrichir et améliorer la qualité de représentation de modèles. Enfin, l'exploration de possibilités d'application dans d'autres domaines de la santé publique et de la recherche en épidémiologie.

BIBLIOGRAPHIE ET WEBOGRAPHIE

Breu, R., Hinkel, U., Hofmann, C., Klein, C., Paech, B., Rumpe, B., Thurner, V., 1997. Towards a formalization of the Unified Modeling Language, in: Akşit, M., Matsuoka, S. (Eds.), ECOOP'97 — Object-Oriented Programming, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 344–366. https://doi.org/10.1007/BFb0053386

Camara, G., 2013. Conception d'un système de veille épidémiologique à base d'ontologies: application à la schistosomiase au Sénégal (Thèse de doctorat). Université Pierre et Marie Curie, Paris) (1971-2017, France, Sénégal.

Carlson, C.J., Boyce, M.R., Dunne, M., Graeden, E., Lin, J., Abdellatif, Y.O., Palys, M.A., Pavez, M., Phelan, A.L., Katz, R., 2023. The World Health Organization's Disease Outbreak News: A retrospective database. PLOS Glob. Public Health 3, e0001083. https://doi.org/10.1371/journal.pgph.0001083

Cissé, P.A., 2016. Simulation à base d'agents de la propagation de la Schistosomiase : une approche de composition et de déploiement de modèles (phdthesis). Université Pierre et Marie Curie - Paris VI.

Cisse, P.A., Camara, G., Dembele, J.M., Lo, M., 2019. An Ontological Model for the Annotation of Infectious Disease Simulation Models, in: International Conference on Innovations and Interdisciplinary Solutions for Underserved Areas. Springer, pp. 82–91.

Cisse, P.A., Dembele, J.M., Lo, M., Cambier, C., 2017. Multi-agent Systems for Epidemiology: Example of an Agent-Based Simulation Platform for Schistosomiasis, in: Bajo, J., Vale, Z., Hallenborg, K., Rocha, A.P., Mathieu, P., Pawlewski, P., Del Val, E., Novais, P., Lopes, F., Duque Méndez, N.D., Julián, V., Holmgren, J. (Eds.), Highlights of Practical Applications of Cyber-Physical Multi-Agent Systems, Communications in Computer and Information Science. Springer International Publishing, pp. 157–168.

COVID-19 Map [WWW Document], n.d. . Johns Hopkins Coronavirus Resour. Cent. URL https://coronavirus.jhu.edu/map.html (accessed 10.17.23).

Cuzin, L., Delpierre, C., 2005. Épidémiologie des maladies infectieuses. EMC - Mal. Infect. 2, 157–162. https://doi.org/10.1016/j.emcmi.2005.10.001

Drogoul, A., Amouroux, E., Caillou, P., Gaudou, B., Grignard, A., Marilleau, N., Taillandier, P., Vavaseur, M., Vo, D.-A., Zucker, J.-D., 2013. GAMA: A Spatially Explicit, Multi-level, Agent-Based Modeling and Simulation Platform, in: Advances on Practical Applications of Agents and Multi-Agent Systems. Presented at the Practical Applications of Agents and Multi-Agent Systems, pp. 271–274.

Dugerdil, P., 2005. Impact des décisions informatiques: introduction à l'informatique pour le décideur non-informaticien. PPUR presses polytechniques.

Europe, W.H.O.R.O. for, 2019. Interview with Professor Christopher J.L. Murray, Director and co-founder of the Institute for Health Metrics and Evaluation (IHME) and the Chair of the Health Metrics Sciences Department at the University of Washington. Public Health Panor. 05, 28–31.

Ferber, J., 1995. Les Systèmes multi-agents: vers une intelligence collective. InterEditions.

Fernandez, M., n.d. Chapitre I – Les ontologies.

Fianyo, Y.E., 2001. Couplage de modèles à l'aide d'agents : le système OSIRIS. Paris 9.

Fontenille, D., Lagneau, C., Lecollinet, S., Lefait Robin, R., Setbon, M., Tirel, B., Yébakima, A. (Eds.), 2013. Modes de transmission des agents infectieux / La dynamique de la maladie versus celle de l'infectiosité, in: La

lutte antivectorielle en France, Expertise collégiale. IRD Éditions, Marseille, pp. 4–7. https://doi.org/10.4000/books.irdeditions.1303

Freeman, Eric, Freeman, Elisabeth, 2010. Design Patterns Tête la première.

Hassas, S., 2017. Unified Modeling Language UML.

Holovaty, A., Kaplan-Moss, J., Gilmore, J., 2008. The definitive guide to Django: Web development done right, Expert's voice in Web development. Apress ; Distributed by Springer-Verlag, Berkeley, Calif. : New York.

Homepage | The Institute for Health Metrics and Evaluation [WWW Document], n.d. URL https://www.healthdata.org/ (accessed 10.17.23).

Howerton, E., Contamin, L., Mullany, L.C., Qin, M., Reich, N.G., Bents, S., Borchering, R.K., Jung, S., Loo, S.L., Smith, C.P., Levander, J., Kerr, J., Espino, J., Panhuis, W.G. van, Hochheiser, H., Galanti, M., Yamana, T., Pei, S., Shaman, J., Rainwater-Lovett, K., Kinsey, M., Tallaksen, K., Wilson, S., Shin, L., Lemaitre, J.C., Kaminsky, J., Hulse, J.D., Lee, E.C., McKee, C., Hill, A., Karlen, D., Chinazzi, M., Davis, J.T., Mu, K., Xiong, X., Piontti, A.P. y, Vespignani, A., Rosenstrom, E.T., Ivy, J.S., Mayorga, M.E., Swann, J.L., España, G., Cavany, S., Moore, S., Perkins, A., Hladish, T., Pillai, A., Toh, K.B., Longini, I., Chen, S., Paul, R., Janies, D., Thill, J.-C., Bouchnita, A., Bi, K., Lachmann, M., Fox, S., Meyers, L.A., Consortium, U.C.-19 M., Srivastava, A., Porebski, P., Venkatramanan, S., Adiga, A., Lewis, B., Klahn, B., Outten, J., Hurt, B., Chen, J., Mortveit, H., Wilson, A., Marathe, M., Hoops, S., Bhattacharya, P., Machi, D., Cadwell, B.L., Healy, J.M., Slayton, R.B., Johansson, M.A., Biggerstaff, M., Truelove, S., Runge, M.C., Shea, K., Viboud, C., Lessler, J., 2023. Informing pandemic response in the face of uncertainty. An evaluation of the U.S. COVID-19 Scenario Modeling Hub. https://doi.org/10.1101/2023.06.28.23291998

Le Moigne, J.-L., 1990. La modélisation des systèmes complexes. Dunod, Paris.

Liu, Z., Magal, P., Seydi, O., Webb, G., 2020. Understanding Unreported Cases in the COVID-19 Epidemic Outbreak in Wuhan, China, and the Importance of Major Public Health Interventions. Biology 9, 50. https://doi.org/10.3390/biology9030050

Marchal, B., 2002. XML by Example. Que Publishing.

Momjian, B., 2001. PostgreSQL: introduction and concepts. Addison-Wesley, Boston, Mass.

OMG, 2017. About the Unified Modeling Language Specification Version 2.5.1 [WWW Document]. URL https://www.omg.org/spec/UML/About-UML/ (accessed 10.26.23).

Siebert, J., 2011. Approche multi-agent pour la multi-modélisation et le couplage de simulations. Application à l'étude des influences entre le fonctionnement des réseaux ambiants et le comportement de leurs utilisateurs. (phdthesis). Université Henri Poincaré - Nancy I.

Treuil, J.-P., Drogoul, A., Zucker, J.-D., 2008. Modélisation et simulation à base d'agents: exemples commentés, outils informatiques et questions théoriques. Dunod, Paris.

What we do [WWW Document], 2010. URL https://www.ecdc.europa.eu/en/about-ecdc/what-we-do (accessed 10.17.23).

Workowski, K.A., 2015. Centers for Disease Control and Prevention Sexually Transmitted Diseases Treatment Guidelines. Clin. Infect. Dis. 61, S759–S762. https://doi.org/10.1093/cid/civ771