ALLEMAND Fabien
Promotion 2024
2021-2022

Diplôme d'ingénieur Télécom Physique Strasbourg
Spécialité Informatique et Réseaux

Mémoire de stage de 1ère année

Conversion de données géométriques en modèles physiques de thermique du bâtiment

Centre de modélisation et de simulation de Strasbourg

Cemosis
7 rue René Descartes
Strasbourg, France
contact@cemosis.fr

DJATOUTI Zohra
zohra.djatouti@cemosis.fr
03 68 85 02 71

Du 07/06/2022 au 15/07/2022

Conversion de données géométriques en modèles physiques de thermique du bâtiment

La consommation énergétique d'un bâtiment a un impact direct sur le coût en énergie et sur l'environnement. La plateforme Ktirio a pour but d'estimer différents paramètres au moyen de simulations qui permettront de valider des plans de construction ou de rénovation.
Afin de réaliser la simulation, il faut créer un fichier ayant le bon format contenant les informations nécessaires: architecture du bâtiment, matériaux, lieu. Ces informations proviennent principalement des fichiers exportés depuis les logiciels de CAO. L'objectif est de pouvoir manipuler de tels fichiers pour y ajouter d'éventuelles informations complémentaires et les convertir en fichiers utilisables pour la simulation sans perdre d'information pour obtenir des résultats précis. L'outil de conversion de fichier doit être suffisamment robuste pour traiter correctement les géométries complexes présentes dans l'architecture des bâtiments et les éventuels défauts de conception dans le logiciel de CAO. Des outils d'analyse doivent être créés afin de contrôler l'intégrité des informations et des tests doivent être mis en place pour vérifier le bon déroulement de la conversion.

Avant d'apporter des modifications à l'outil open-source utilisé pour convertir les fichiers, il faut s'assurer que l'installation de tous les outils est correcte grâce à des tests automatiques et des fichiers de référence.
Un pré-traitement des fichiers exportés depuis les logiciels de CAO est utile pour ajouter les propriétés thermiques des matériaux. Il faut alors créer de nouveaux fichiers de référence.
Différents outils de diagnostics sont utiles pour vérifier le bon fonctionnement de la conversion. Ils ont pour but de faciliter l'accès aux informations contenues dans les fichiers complexes issus des logiciels de CAO. Il est alors possible d'analyser rapidement le contenu d'un fichier et de le comparer avec un fichier exporté différemment ou un fichier converti.

Développer un outil de conversion de fichiers contenant des structures géométriques complexes ainsi que des propriétés physiques importantes est un travail de recherche où la méthode, la rigueur et la maîtrise des outils à disposition sont nécessaires.

Converting geometric data into physical building thermal models

Proper energy managment in a building can lead to less energy cost, fewer impact on the environnement and an overall improved confort. Thermal simulation can be used to study a building before its construction or renovation.
Within Ktirio Project, building's architecture, materials and location have to be stored in a specific structured file in order to perform simulation. These data are exported from CAD software in large structured files.
The goal is to develop a tool able to analyse, modify (adding additional data) and convert these files into files usable for accurate simulation, without loosing or overlooking any data. The tool has to be reliable enough to support building's complex geometry data and potential design flaws in CAD software. Analysis tools need to be created in order to control data integrity and various tests have to be set up to ensure the accuracy of the conversion.

Before any modification to the open-source tool used to convert files, a serie of tests on multiple reference files have to be performed to ensure the installation of all required tools was successful.
Files exported from the CAD software can be pre-processed and additional data such as materials' thermal properties can be inserted. At this stage a new set of reference files is created.
Analysis tools can be used to check if the conversion was successful. They have to facilitate access to data contained in complex files and allow to perform quick and precise comparison between files, either CAD files exported differently or a CAD file and the result of the conversion.

Developping tools to analyse and convert files containing highly detailed geometric structure and their thermal porperties is a long process that require methodology and mastering a wide set of tools.

Remerciements

Je tiens à remercier toutes les personnes qui m'ont accompagné et aidé tout au long du stage et lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements à mon maître de stage, Mme Zohra DJATOUTI, ingénieur-chercheur au sein de Cemosis, pour son accueil, le temps passé ensemble et le partage de son expertise. Son accompagnement et ses explications m'ont permis de mieux appréhender les missions qui m'ont été confiées mais aussi d'acquérir de nouvelles compétences.

Je remercie également les élèves du master CSMI de l'UFR de mathématique et d'informatique de l'université de Strasbourg en stage à Cemosis pour leur bonne humeur et leur esprit d'entre aide.

Table des matières

Liste des figures

1 Introduction

1.1 Cemosis

Le Centre de Modelisation et de Simulation de Strasbourg (*Cemosis*)[3], fondé en 2013 par Christophe Prud'homme, est spécialisé dans les domaines de la modélisation, la simulation, l'optimisation et le calcul de haute performance. Hébergé par l'Institut de Recherche Mathématique Avancée (*IRMA*), Cemosis a été créé dans le but de répondre à des problématiques multidisciplinaires à l'aide d'outils de simulation et de modélisation, de proposer des formations ainsi que d'établir un lien entre acteurs de la recherche, PME/PMI et grandes entreprises de la région Alsace.[4, 5, 24]

Cette plateforme de collaboration, regroupant moins de cinquante employés répartis dans les locaux de l'UFR (Unité de Formation et de Recherche) de mathématique et d'informatique de Strasbourg, s'engage dans différents types de projets:

- Projets à court terme utilisant principalement des outils déjà développés.
- Projets nécessitant le développement ou l'adaptation d'outils pour traiter des modèles existants, réalisés par un ingénieur ou un postdoctorant recruté pour une durée allant jusqu'à deux ans.
- Projets d'une durée minimale de trois ans, généralement associés à une convention CIFRE (Convention Industrielle de Formation par la Recherche[25]), ayant pour but de créer les modèles et les logiciles nécessaires.

Les projets actuellement en développement sont majoritairement liés au domaine de la santé (AngioTK, Vivabrain, Eye2Brain) et de l'énergie (Tonus, Ibat).[6]

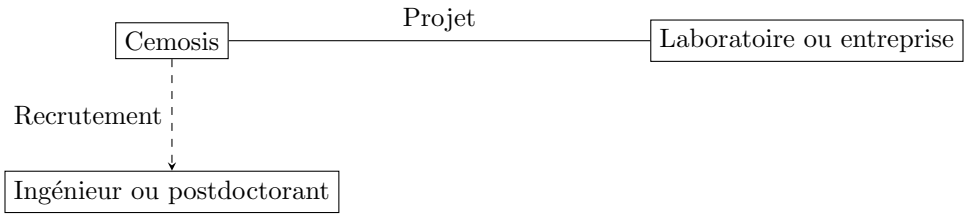

Figure 1: Organigramme d'un projet Cemosis

1.2 Contexte

En 2021, près de la moitié de l'énergie consommée en France était liée au secteur du bâtiment, émettant plus de 123 millions de tonnes de CO2 [23]. De nombreux bâtiments souffrent d'une mauvaise isolation ou d'un système de chauffage/rafraîchissement inadapté engendrant des pertes énergétiques. Afin de limiter le gaspillage il faut concevoir ou rénover un bâtiment dans le but de minimiser les pertes.
Réaliser une simulation en prenant en compte l'architecture (forme du bâtiment, taille des pièces, etc.), les matériaux (isolation, ouvertures, etc.) et l'emplacement (exposition au soleil, climat, etc.) permet d'estimer les perfomances énergétiques du bâtiment.

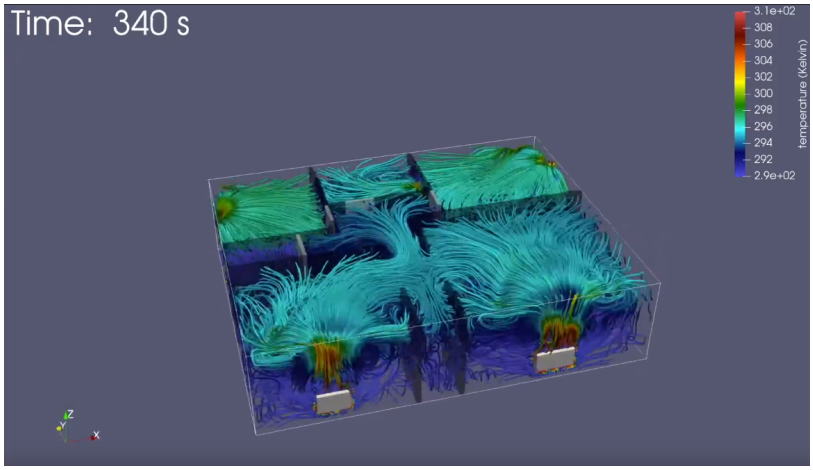

Figure 2: Simulation thermique d'un bâtiment par Cemosis [26]

1.3 Projet Ktirio

Le projet Ktirio a pour objectif de créer une plateforme en ligne permettant le suivi en temps réel, la simulation et la prédiction à court, moyen et long terme ainsi que l'analyse du comportement thermique, du confort et de la performance énergétique de bâtiments.
La plateforme permettra de suivre et de simuler le comportement thermique d'un bâtiment. Pour cela, elle intègre des outils qui permettront de faciliter et robustifier les différentes étapes d'une analyse énergétique, allant de la génération automatique de modèles à partir de la géométrie 3D à la production d'indicateurs de confort et de performance utiles à la prise de décision quant à la gestion et la rénovation du bâtiment.

Cet outil interactif permettra de visualiser de façon précise un bâtiment et ses composants (position, caractéristiques et propriétés thermiques des éléments de construction), de définir les zones thermiques (regroupements de pièces ayant les mêmes propriétés thermiques) et d'effectuer des simulations sur un bâtiment.
Grâce à un ensemble de capteurs répartis dans la construction, il sera aussi possible de suivre en temps réel l'évolution de nombreux paramètres thermiques et de confort.[24]

Les données géométriques et les propriétés thermiques (lorsqu'elles sont renseignées) du bâtiment sont automatiquement extraites du fichier exporté par un logiciel de CAO et tansmises à un générateur automatique de modèle qui permet de créer les fichiers nécessaires à la simulation. Si les propriétés thermiques ne sont pas renseignées dans le fichier issu du logiciel de CAO alors elles doivent être importées avant de réaliser une simulation.

Le logiciel de CAO utilisé doit offrir la possibilité d'exporter au format IFC. Ce format fait partie du BIM, un ensemble de normes internationales, européennes et françaises qui sont de plus en plus répandues dans les domaines du génie civil, des travaux publics, des infrastructures et des réseaux. (voir section 3.1, page 6)

1.4 Objectifs

L'objectif est d'assurer le bon traitement des informations afin d'obtenir des résultats de simulations corrects.
Pour cela, il faut:

- Développer des outils permettant d'analyser les fichiers sources.
- Accélérer et fiabiliser la saisie des données géométriques et des propriétés thermiques qui peuvent être laborieuses et sujettes à erreur.
- Étudier et améliorer le générateur automatique de modèles.
- Développer des outils permettant d'analyser les fichiers créés.
- Créer des tests permettant de vérifier le bon fonctionnement des modèles pour la simulation.

1.5 Organisation

Le projet est entièrement géré sur GitHub, ce qui offre de nombreux avantages. La gestion des versions et le travail en équipe est simplifié grâce au travail en ligne. La mise en évidence et la gestion des modifications à apporter est rendue efficace grâce aux Issues et aux Actions. La gestion globale du projet peut être faite en temps réel et de façon collective au moyen de l'onglet Projects.

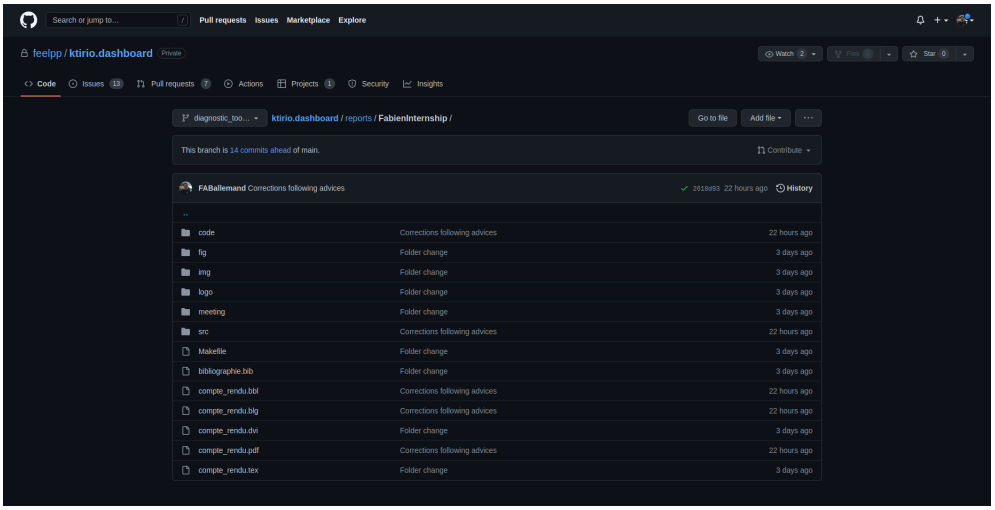

Figure 3: Onglet Code de l'interface en ligne de GitHub

Le framework Pytest (voir section 3.4, page 7) est configuré dans le fichier `pytest.ini` dans le répertoire principal. Ainsi, les tests Python ayant la bonne syntaxe sont exécutés automatiquement sur GitHub grâce à l'intégration continue mise en place dans les fichiers `.github/workflows/ci.yml`.
Les résultats des tests sont visibles dans la section Actions du panneau de contrôle GitHub.

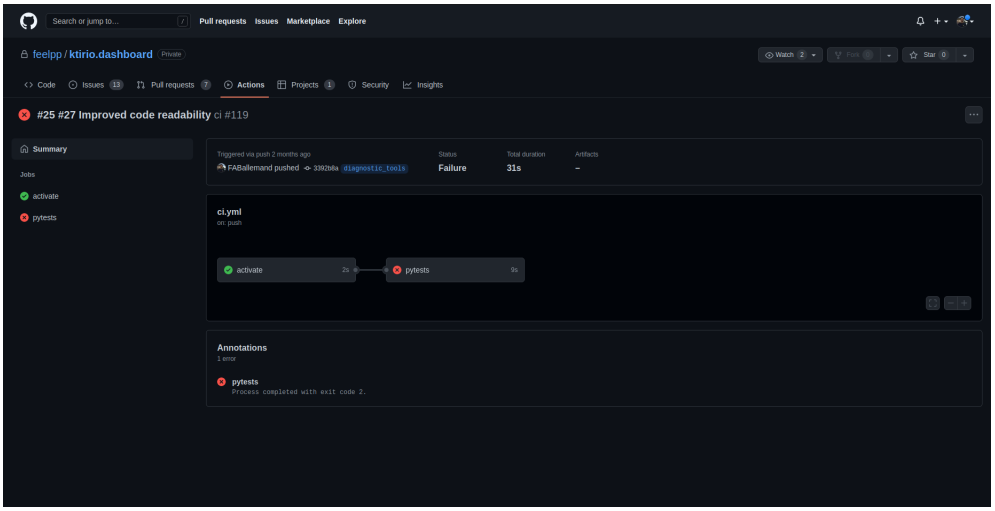

Figure 4: Onglet Actions de l'interface en ligne de GitHub

L'organisation au sein de l'équipe repose sur une bonne communication au quotidien grâce à des rencontres dans les bureaux ou des discussions via Slack (voir section 3.8, page 8).
Une réunion Zoom (voir section 3.8, page 8) a lieu chaque lundi matin afin d'organiser la semaine, présenter les derniers résultats et répondre aux questions.

2 Présentation du projet

Les données relatives au bâtiments peuvent provenir de différentes sources. L'outil permettant la simulation nécessite des fichiers contenant toutes les informations utiles dans un format prédéfini. Le but est de collecter toutes les informations disponibles dans le fichier source et de les transférer dans un fichier utilisable pour la simulation.

2.1 Conversion de fichiers

De nombreux facteurs sont importants lors de la simulation énergétique d'un bâtiment:

- Architecture du bâtiment (dimensions, formes, liens entre les différentes zones).
- Propriétés thermiques des matériaux.
- Localisation (climat et exposition au soleil).
- Système de chauffage/rafraîchissement.

Pour récupérer les informations sur l'architecture, il est possible d'utiliser les plans créés par l'architecte dans un logiciel spécialisé comme Revit ou ArchiCAD. Les plans sont récupérés au format IFC: un format de fichier standardisé afin de pouvoir échanger des informations entre logiciels.
Remarque || Les fichiers utilisés dans le projet suivent la norme IFC2X3.
Cependant la simulation finale n'est pas exécutée sur les fichiers IFC. Il faut convertir les fichier IFC en fichier FMU.

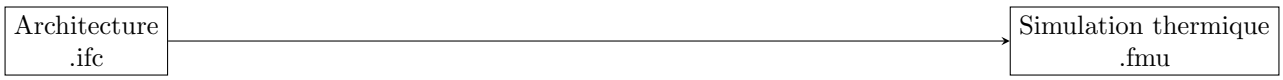

Figure 5: Objectif de conversion

La conversion se déroule en deux étapes:

- Conversion du fichier IFC en fichier Modelica.
- Conversion du fichier Modelica en fichier FMU.

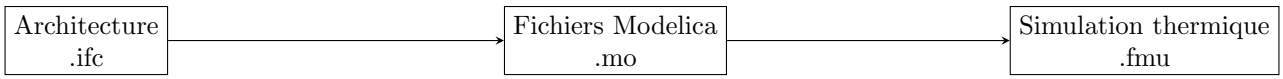

Figure 6: Conversion en deux étapes

La première partie utilise l'outil de génération automatique de modèles à partir de *templates* CoTeTo et la librairie Modelica BuildingSystems. Pendant cette étape l'entièreté de la structure du bâtiment est transférée dans un fichier Modelica.
La deuxième étape est réalisée par le logiciel Dymola.

2.2 Détails de la conversion en fichier Modelica

Les fichiers IFC ont une sctructure particulière qui respecte certaines normes. La librairie Python IfcOpenShell permet d'analyser ces fichiers de façon efficace.
La librairie Modelica BuildingSystems définit les objets nécessaires pour décrire un bâtiment dans Modelica. Cette librairie permet de réaliser des simulations énergétiques, notamment sur les modèles multizones utilisés dans Ktirio.
BIM2Modelica est un template pour générer automatiquement des modèles suivant les spécifications de BuildingSystems. BIM2Modelica repose sur l'outil open-source CoTeTo.

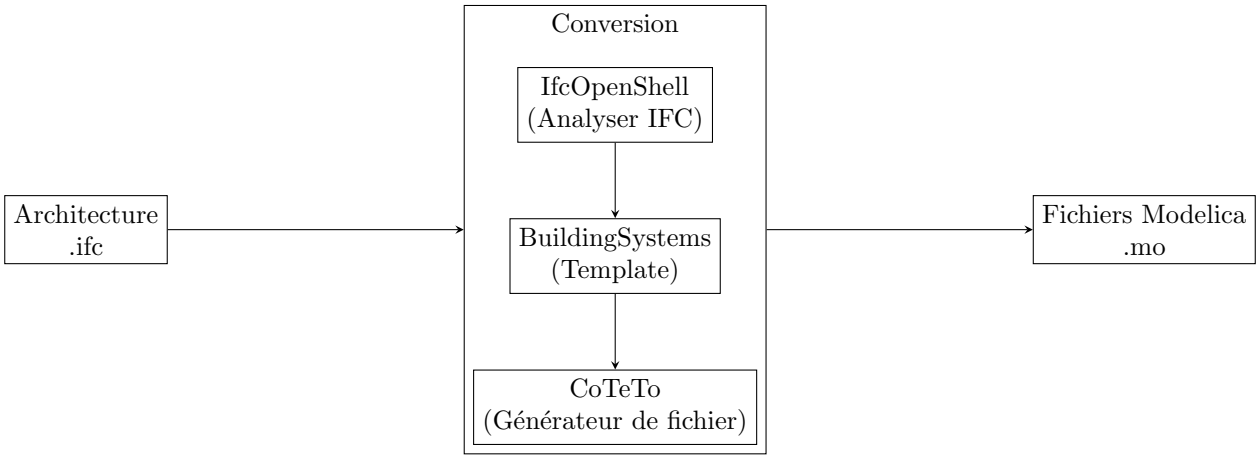

Figure 7: Conversion des fichiers IFC en fichiers Modelica

2.3 Matériaux et propriétés thermiques

Les propriétés thermiques des matériaux n'étant pas toujours contenues dans les fichiers IFC utilisés, il faut ajouter une étape de pré-traitement des fichiers IFC pour y ajouter ces propriétés.

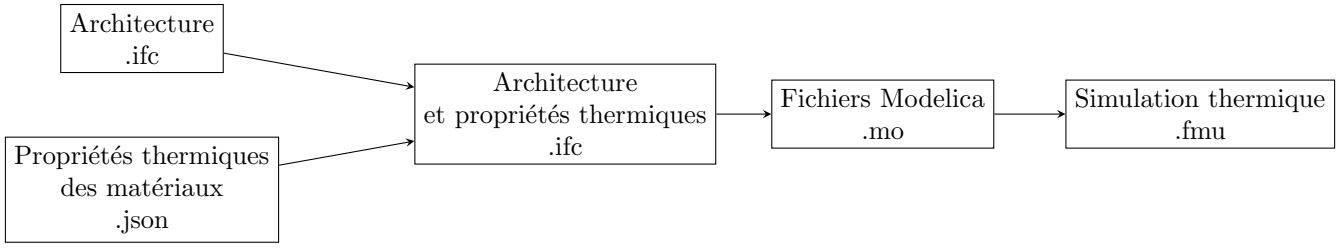

Figure 8: Pré-traitement et conversion des fichiers IFC

2.4 Géométrie et éléments de construction

Il existe différents logiciels de CAO utilisés pour concevoir des bâtiments. Chaque logicel présente des fonctionnalités différentes pour la conception et pour l'exportation. Les fichiers IFC obtenus ne contiennent donc pas systématiquement les mêmes données et les mêmes structures. Il est aussi possible de rencontrer des erreurs de conception dans les fichiers exportés (collision entre deux éléments, mauvaise définition des zones thermiques).
Il faut donc être capable de choisir les bons paramètres d'exportation et de corriger les erreurs de conceptions lors de la conversion.

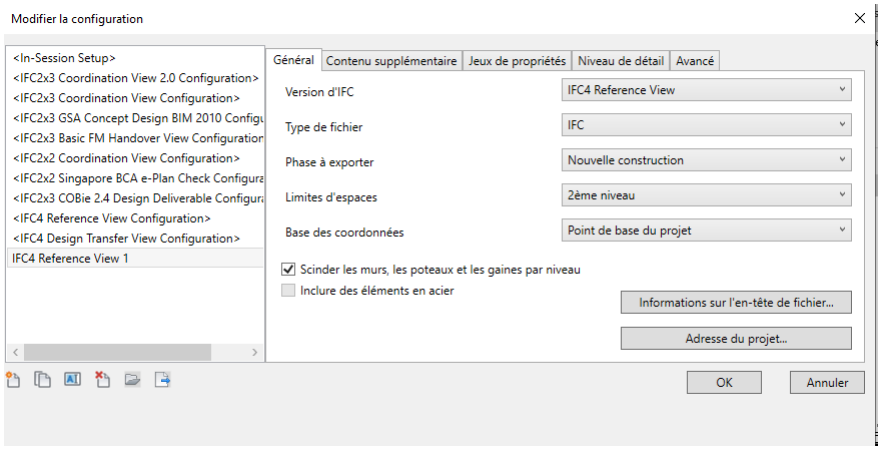

Figure 9: Fenêtre de paramètre d'exportation au format IFC dans le logiciel Revit

3 Outils

3.1 Normes et standards

BIM:
Building Information Modeling (BIM) est le processus global de création et de gestion des informations pour une ressource de construction. Le BIM intègre des données structurées et pluridisciplinaires pour produire la représentation numérique d'une ressource tout au long de son cycle de vie, de la planification à la conception et de la construction à l'exploitation.
La norme ISO 19650-1 donne la définition suivante:
"utilisation d'une représentation numérique partagée d'un actif bâti pour faciliter les processus de conception, de construction et d'exploitation et former une base fiable permettant les prises de décision."

IFC:
Norme: IFC2X3
Le format IFC (Industry Foundation Classes) est un format de fichier standardisé (norme ISO 16739) orienté objet utilisé par l'industrie du bâtiment pour échanger et partager des informations entre logiciels.

3.2 Langages de programmation

Python:
Version: 3.8.13
Python[14] est un langage de programmation interprété, multi-paradigme et multiplateformes valorisant la programmation impérative structurée, fonctionnelle et orientée objet.

Modelica:
Modelica[11] est un langage de modélisation pluridisciplinaire qui supporte différents types de formalisme. Modelica peut être utilisé pour modéliser des systèmes discrets et continus de différentes natures à l'aide des librairies standard multi-domaines.

3.3 Environnement Python

Anaconda:
Anaconda[1] est un gestionnaire open-source de paquets et d'environnements, disponible sur Windows, macOS, and Linux pour le langage Python en particulier.

3.4 Bibliotèques Python

IfcOpenShell:
Version: 0.6.0b0
IfcOpenShell[10] est une librairie open source qui permet de manipuler des fichiers au format IFC.
IfcOpenShell utilise Open CASCADE pour convertir la géométrie des fichiers IFC dans un format utilisable par tous les logiciels de CAO.

PythonOCC:
Version: 7.5.1
PythonOCC[12] est un wrapper Python pour OpenCASCADE C++.
Open CASCADE Technology (OCCT) est une librairie de géométrie 3D open source qui permet de modéliser et visualiser des esquisses géométriques, des surfaces et des solides tout en assurant une large compatibilité de formats de fichiers pris en charge.

Pytest:
Version: 7.1.2
Pytest[13] est un framework de test logiciel basé sur le langage de programmation Python. Il peut être utilisé pour écrire divers types de tests logiciels, notamment des tests unitaires, des tests d'intégration, des tests de bout en bout et des tests fonctionnels.

Xlsxwriter:
Version: 3.0.3
Xlsxwriter[21] est un module Python permettant de créer des fichiers au format xlsx.

3.5 Bibliotèques Modelica

BuildingSystems:
BuildingSystems[2] est une librairie open-source développée pour la simulation énergétique de bâtiments à différentes échelles (pièces, bâtiments, quartiers...).
Cette librairie permet de réaliser des simulations énergétiques sur les modèles multizones utilisés dans Ktirio.

3.6 Outils de CAO

Solibri:
Solibri[17] Anywhere est un outil permettant de visualiser les fichiers IFC. Son utilisation se veut simple, rapide et efficace pour les professionnels qui recherchent à trouver des informations précises et pertinentes sur le bâtiment en construction.

Revit:
Revit[15] est un logiciel de conception de bâtiment édité par la société Autodesk qui permet de dessiner un modèle en 3D d'un bâtiment afin de créer divers documents nécessaires à sa construction (plans, perspectives, ...).
Revit a la particularité d'être un logiciel BIM multi-métiers destiné aux professionnels du BTP (ingénieurs, architectes, dessinateurs-projeteurs, entrepreneurs, ...).
Les fichiers au format IFC sont nativement supportés par Revit. Il est possible de visualiser le contenu du fichier de différentes façons (2D, 3D, coupes, filtres...) et d'y apporter des modifications.

3.7 Outil de simulation

Dymola:
Dymola[8] (Dynamic Modeling Laboratory) est un outil de modélisation et de simulation de systèmes intégrés et complexes, destiné à être utilisé dans les secteurs de l'automobile, l'aéronautique, la robotique, les processus et d'autres applications.
Dymola est basé sur le langage de modélisation Modelica.

3.8 Utilitaires

CoTeTo:
Code Templating Tool[7] (CoTeTo) est un outil pour la génération de code source ou texte provenant de sources différentes à l'aide de *templates*. Cet outil fonctionnant par interface graphique, ligne de commande ou module Python peut être modifié et amélioré pour satisfaire les besoins d'un projet.
L'outil Bim2Modelica développé par Cemosis est un template pour générer automatiquement des modèles suivant les spécifications de BuildingSystems.

GitHub:
GitHub[9] est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

Slack:
Slack[16] est une application de messagerie pour les entreprises. La communication repose sur un système d'espaces de travail dédiés appelés canaux, qui rassemblent les personnes et les informations pertinentes.

Zoom:
Zoom[22] Vidéo Communications est un service de conférence à distance qui combine la vidéoconférence, les réunions en ligne, le chat et la collaboration mobile.

4 Vérification de l'installation

La première étape pour commencer le projet consiste à installer et vérifier la bonne installation des divers outils.

4.1 Génération de fichiers

Une fois les outils installés (voir section 8, page 22), il faut vérifier leur bon fonctionnement. Pour cela, deux jeux de fichiers de référence sont donnés dans le dépôt BIM2Modelica: `UdKb_Unit_Test_Cases` et `SBT_Unit_Test_Cases`. Dans `IFC/IFC2X3`, ce sont les fichiers au format IFC. Ces fichiers ont été convertis en fichiers Modelica avec BIM2Modelica dans `ModelicaModels/IFC2X3`. Pour vérifier l'installation, il faut effectuer la conversion des fichiers IFC donnés et comparer les résultats aux fichiers Modelica donnés.

Script de génération: Le script Python `BIM2Modelica/tests/generate_reference.py` permet de générer ces fichiers dans `BIM2Modelica/tests/ModelicaModels_reference` avec les options souhaitées. La fonction config permet de créer une copie du fichier `Package.inf` dans un répertoire temporaire en modifiant les paramètres nécessaires. Il faut ensuite spécifier à l'outil de génération automatique de modèles d'utiliser ce fichier lors de son initialisation de la façon suivante:

```
gen.init(packagePath = os.path.realpath(os.path.dirname(__file__)) + "/tmp")
```

Script de test On peut aussi créer un script qui permet de tester si de nouveaux fichiers générés coïncident avec les fichiers de référence. De cette façon, il est possible de repérer s'il y a des différences entre les fichiers de référence et les fichiers générés, révélant une mauvaise installation.

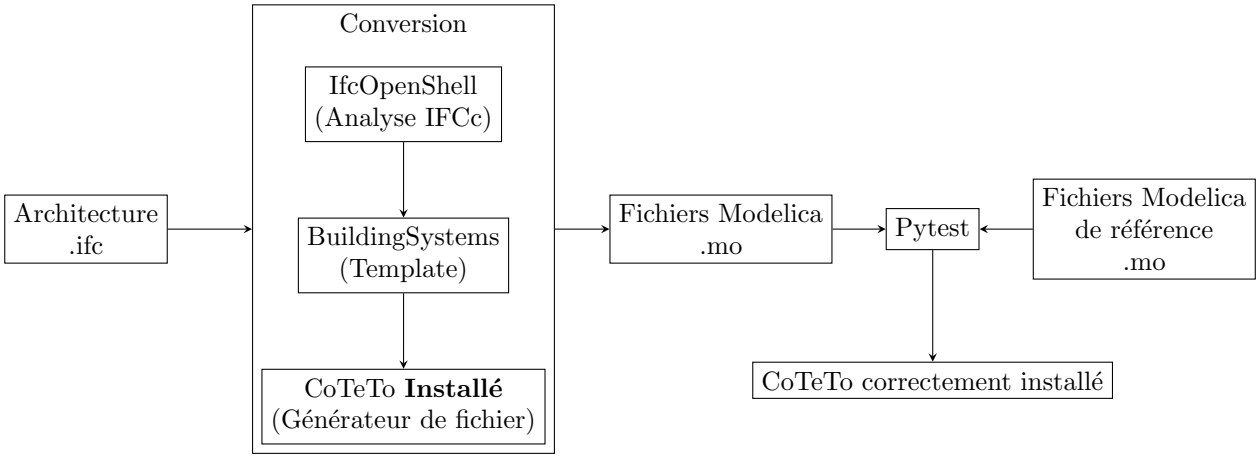

Figure 10: Protocole de vérification des fichiers générés

Le pytest `BIM2Modelica/tests/check_ifc2Modelica.py` génère de nouveaux fichiers dans un dossier temporaire de façon similaire au script `BIM2Modelica/tests/generate_reference.py` puis compare chaque fichier généré avec le fichier référence qui lui est associé grâce à la fonction compareFiles. Cette fonction compare le contenu des deux fichiers ligne par ligne à l'exception de la quatrième ligne qui est ignorée car elle ne contient pas d'informations importantes si ce n'est la date de génération du fichier. Ci-dessous la boucle principale de la fonction compareFiles:

```
with open("tmp/" + mo_files_list[i], 'r') as generated, open(mo_expected_files_list[i],
    'r') as expected:
    generated_content = generated.readlines()
    expected_content = expected.readlines()
    for j in range(len(generated_content)):
        if  generated_content[j] != expected_content[j] and j != 3:
            print("tmp/" + mo_files_list[i] + " and " + mo_expected_files_list[i] + "
  are different line ", j)
            return False
```

Étude d'un fichier incorrect: Lors des premiers tests, un des fichiers de référence (`BIM2Modelica/IFC/IFC2X3/UdKB_Unit_Test_Cases/RooftopBuilding4Zones_Thin.ifc`) engendrait une erreur de segmentation.
À ce stade, une erreur provient soit de l'outil (erreur lors du développement, incompatibilité suite à une mise à jour, etc.) soit de l'installation de l'outil. Il faut donc comprendre d'où provient l'erreur pour s'assurer qu'elle ne provient pas de l'installation.

Débogage: Une première approche pour localiser l'erreur consiste à ajouter des messages à afficher lors de l'exécution du programme. Lors de l'analyse du fichier on peut notamment afficher les informations de chaque objet traité grâce à la méthode .get_info() dans la fonction mapIFCtoBuildingDataModel dans le fichier `BIM2Modelica/CoTeTo/Generators/IFC_MultiZoneBuildings_Modelica/Filters/IfcLib/Filter01.py`.
En ajoutant quelques print de débogage, on remarque que le programme s'arrête lors de la création du maillage d'un mur ayant les informations suivantes:

```
{'id': 75381, 'type': 'IfcWallStandardCase', 'GlobalId': '3RB9qJdj0cJPxaRd_LTXkx', '
    OwnerHistory': #12=IfcOwnerHistory(#7,#11,$,.ADDED.,$,$,$,1493320357), 'Name': '
    WALL', 'Description': None, 'ObjectType': None, 'ObjectPlacement': #75340=
    IfcLocalPlacement(#99,#75339), 'Representation': #75377=IfcProductDefinitionShape($
    ,$,(#75365,#75374)), 'Tag': 'DB2C9D13-9ED0-264D-9EE4-6E7F95761BBB'}
```

Observation avec Solibri: Les informations obtenues contiennent l'ID de l'objet. En ouvrant le fichier IFC avec Solibri et en utilisant la barre de recherche, on peut retrouver l'objet qui engendre l'erreur.
On s'aperçoit alors que l'objet (nommé Mur.0.2 dans le logiciel) n'est non seulement pas visible mais possède des paramètres incohérents (surfaces nulles), paramètres qui sont cohérents sur d'autres objets comme le Mur.0.15 (ce qui élimine l'hypothèse d'une erreur liée à Solibri). En parcourant les autres objets, on s'aperçoit qu'il existe un autre mur (Mur.0.8) qui présente les mêmes caractéristiques.

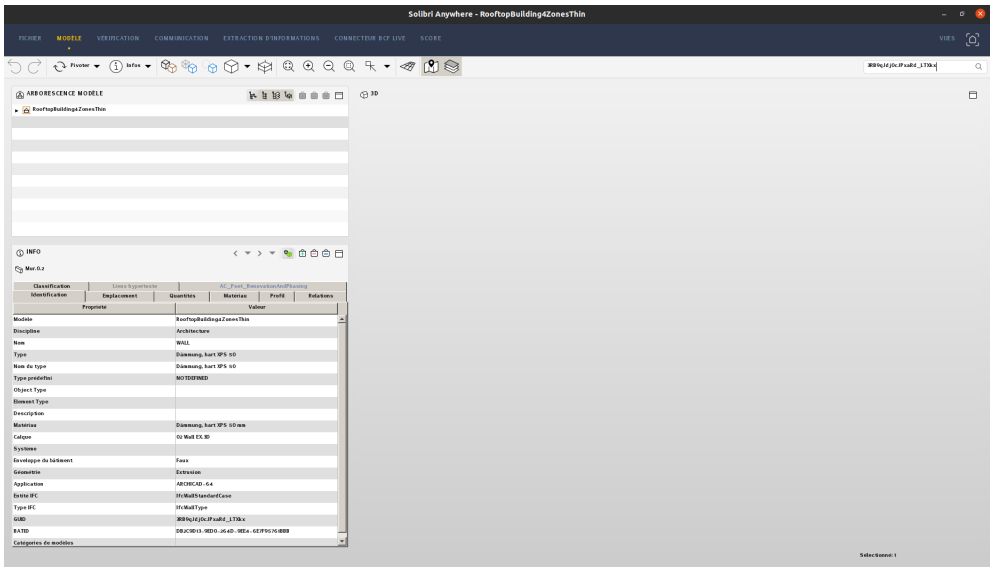

Figure 11: Visualisation de l'objet mal défini avec Solibri

Etude du fichier IFC: Pour comprendre d'où vient l'erreur il faut donc étudier les fichiers IFC. La méthode consiste à étudier la définition dans le fichier IFC d'un objet bien défini (par exemple le Mur.0.15 ayant l'ID 0V6E6ruyWeGh4ngRQrNzjX) pour comprendre comment sont enregistrés les paramètres et caractéristiques. Puis de la comparer avec la définition du Mur.0.2 qui génère l'erreur.
Les définitions qui semblent importantes sont ObjectPlacement et Representation. En partant de ces deux points de référence, il faut explorer l'arbre en suivant les #XXXX et comprendre ce que représente chaque branche et feuille à l'aide de la documentation du format IFC2X3.
Il est possible de faire de même pour les matériaux.

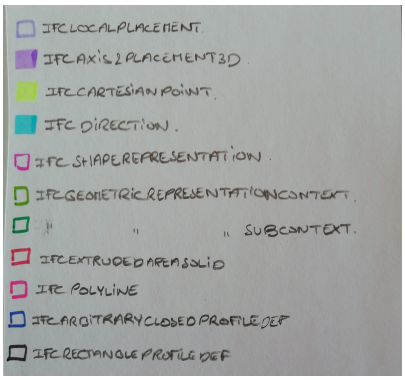

Figure 12: Légende utilisée dans les arbres IFC

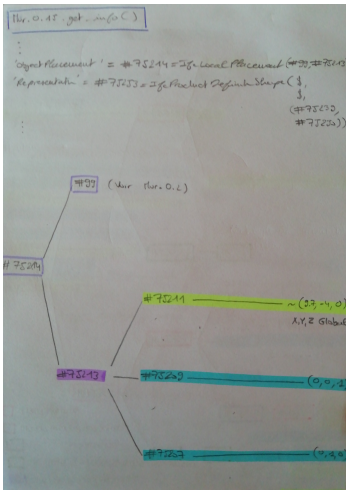

Figure 13: Arbre IFC du placement de l'objet Mur.0.15

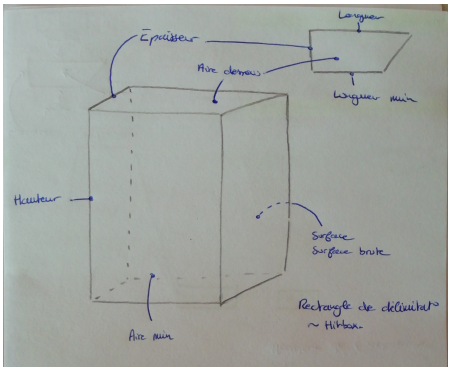

Figure 14: Schema du Mur.0.15 avec paramètres Solibri

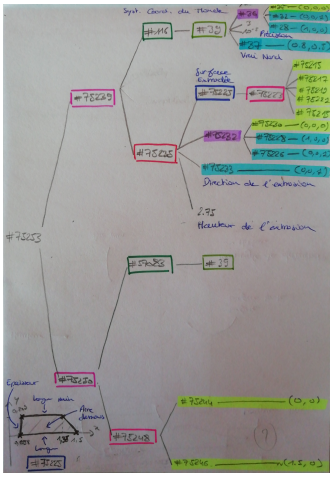

Figure 15: Arbre IFC de la représentation de l'objet Mur.0.15

Avec un peu de patience et de méthodologie, on remarque que le Mur.0.2 est le support d'une porte qui fait exactement la taille du mur. Le mur a donc une surface et un volume nuls ce qui rend le maillage impossible. De même pour le Mur.0.8.
Cette procédure a permis de mettre en évidence des erreurs dans le fichier `RooftopBuilding4Zones_Thin.ifc` qui ne peut donc plus faire office de fichier de référence. Cette hypothèse est confirmée par le fait que le fichier a été commenté par l'auteur de l'outil dans le script pour générer les fichiers de référence.
On peut donc conclure que l'installation des bibliotèques et de l'outil CoTeTo est correcte et peut être utilisée pour la suite du projet.

4.2 Simulation avec Dymola

De premières simulations peuvent être faites avec les fichiers Modelica générés grâce au logiciel Dymola notamment pour vérifier si les fichiers de référence créés précédemment sont corrects.

Script de simulation: Il est possible de créer des scripts pour automatiser les simulations. Un script doit être contenu dans un fichier ayant l'extension .mos. Le script pour lancer la simulation d'un fichier de `UdKB_Unit_Test_Cases` est donné ci-dessous:

```
  Advanced    .Define.DAEsolver = false;
cd("ModelicaModels_reference/IFC2X3/UdKB_Unit_Test_Cases");
simulateModel("OneZone", stopTime=31536000, method="dassl", resultFile="Simulation/
    OneZone");
```

5 Matériaux et propriétés thermiques

Sachant que tous les outils fonctionnent, le but est désormais d'apporter des améliorations au projet existant. Afin d'effectuer les simulations les plus correctes possibles, il faut avoir accès aux propriétés thermiques du bâtiment.

5.1 Pré-traitement d'un fichier

Il est souhaitable d'utiliser les propriétés thermiques des matériaux utilisés dans la construction. Ces propriétés n'étant pas toujours disponibles dans les fichiers IFC (Revit ne permet pas d'exporter le modèle du bâtiment au format IFC avec les propriétés thermiques), il faut les y ajouter "manuellement". Une nouvelle fonction provenant de `ifc_tools.py` est utilisée à cet effet: add_materials_properties permet d'ajouter des propriétés thermiques enregistrées dans un fichier .json.

```
def add_materials_properties(ifc_file, materials_file=None, overwrite=False, output=
    None):
    # Open IFC file
    ifc = ifcopenshell.open(ifc_file)
    # Collect material properties stored in a json file
    with open(materials_file, "r") as read_file:
        materials = json.load(read_file)
    # Add properties to materials
    for mat in ifc.by_type("IfcMaterial"):
        if mat.Name in materials:
            # Specific heat capacity
            Cp = convert_float_or_none(materials[mat.Name].get("Cp"))
            # Boiling point
            Bp = convert_float_or_none(materials[mat.Name].get("Bp"))
            # Freezing point
            Fp = convert_float_or_none(materials[mat.Name].get("Fp"))
            # Conductivity
            k = convert_float_or_none(materials[mat.Name].get("k"))

            ifc.createIfcThermalMaterialProperties(mat, Cp, Bp, Fp, k)
    # Write result (new file, overwrite or default file)
    if output:
        ifc.write(output)
    elif overwrite:
        ifc.write(ifc_file)
    else:
        basename, ext = os.path.splitext(ifc_file)
        ifc.write(basename+"-with-materials"+ext)
```

Figure 16: Version simplifiée de la fonction add_materials_properties

5.2 Création de fichiers de référence

Script de génération: Une fois que les résultats du pré-traitement correspondent à nos attentes, il faut créer des fichiers de référence. Sachant que l'installation des outils et les fichiers utilisés sont corrects, on peut nous même créer les fichiers de référence. Le script Python `BIM2Modelica/tests/generate_reference.py` permet de générer ces fichiers dans `BIM2Modelica/tests/ModelicaModels_reference`. Il fonctionne essentiellement de la même façon que le précédent à la seule différence de la gestion des matériaux grâce à la fonction add_materials_properties avant le traitement du fichier. Le fichier .json contenant les matériaux est créé avant de générer les fichiers à l'aide de la fonction createMaterials qui relève tous les matériaux utilisés dans les fichiers à générer et leur associe des valeurs. Les resultats de la simulation n'étant pas utilisés à cette étape, des valeurs par défaut sont attribuées à chaque matériau.

```
def createMaterials(self, ifc_list_files, path):
    # Create a dictionnary to store materials
    materials = {}
    # Create a dictionnary containing default values for parameters
```

```
    default_values = {"k":"1.5", "rho":"2300", "Cp":"850"}
    # Loop over ifc files
    for file in ifc_list_files:
        # Open file
        ifc_file = ifcopenshell.open(parent_folder + "/" + file[0])
        # Search for materials
        all_materials = ifc_file.by_type("IfcMaterial")
        for m in all_materials:
            print(m.get_info())
            materials[m.Name] = default_values
    # Save information in a json file
    with open(path + "/materials.json", "w") as f:
        json.dump(materials, f, indent=4)
```

Remarque Il est envisageable de développer une interface permettant à l'utilisateur de saisir les valeurs des différentes propriétés thermiques (par exemple dans l'interface graphique interactive du projet Ktirio), voire un système de base de donnée et d'intelligence artificielle qui attriburait les valeurs en fonction du nom des matériaux.

Script de test: On peut aussi créer un script qui permet de tester si les nouveaux fichiers générés coïncident avec les fichiers de référence précédemment créés. De cette façon, il est possible de repérer si de nouvelles modifications dans l'outil de génération automatique de modèles altèrent les fonctionnalités concernant l'ajout des matériaux.

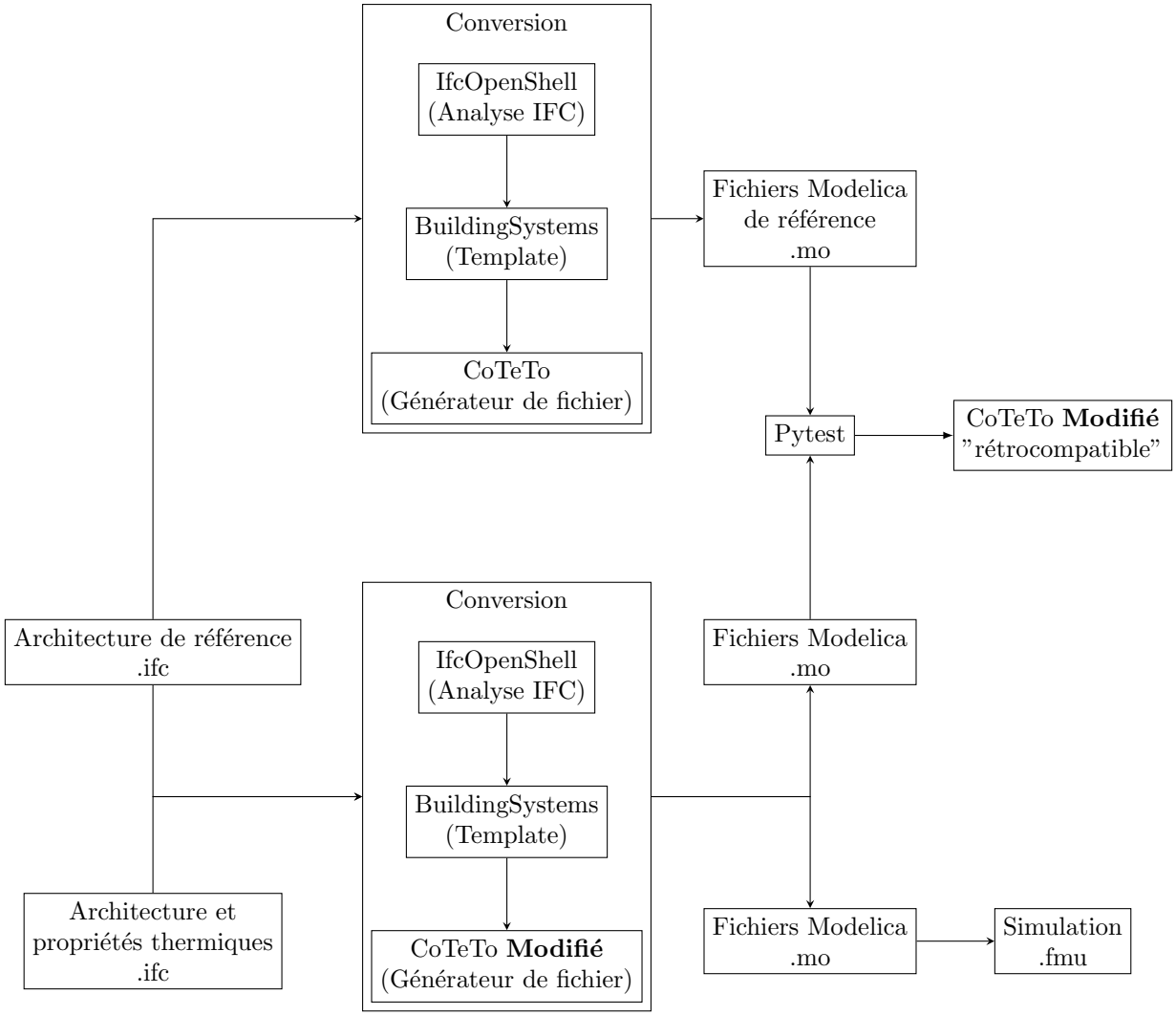

Figure 17: Protocole de génération des fichiers

Le script `BIM2Modelica/tests/check_ifc2Modelica.py` génère de nouveaux fichiers dans un dossier temporaire puis compare chaque fichier généré avec le fichier référence qui lui est associé grâce à la fonction compareFiles.

Script de simulation: Le script de simulation précédent peut être facilement adapté pour fonctionner avec les nouveaux fichiers.

6 Outils d'analyse et de diagnostic

Les matériaux sont désormais traités correctement. Cependant, un nouveau jeu de fichiers IFC révèle de nouvelles failles et de possibles améliorations.

6.1 Conversion et simulation des fichiers

Script Python: La première étape pour tester cet ensemble de fichiers est la conversion de IFC vers Modelica. Pour cela, on utilise un script Python très similaire au script qui génère les fichiers de référence.
À ce stade la fonction config du module `test_util.py` doit être adaptée afin de créer une copie du fichier `Package.inf` en modifiant non seulement ModelicaLibPath et ModelicaMosPath mais aussi l'option calcHygroThermal. Cette option, activée par défaut, doit être mise en position off. Sans cette modification, la majorité des fichiers Modelica générés ne peuvent pas être simulés dans Dymola.

```
def config(ModelicaLibPath, ModelicaMosPath, outputfile="tmp/Package.inf",
    calcHygroThermal="off"):
     ...
     if calcHygroThermal == "on":
         for line in fileinput.input([configfile], inplace=True):
             if line.strip().startswith("calcHygroThermal"):
                 line = "calcHygroThermal = on\n"
             sys.stdout.write (line)
     elif calcHygroThermal == "off":
         for line in fileinput.input([configfile], inplace=True):
             if line.strip().startswith("calcHygroThermal"):
                 line = "calcHygroThermal = off\n"
             sys.stdout.write (line)
     else:
         print("'" + calcHygroThermal + "'" +
               " is not an option for calcHygroThermal. Default option 'on' is used
    instead.")
```

Figure 18: Extrait de la fonction config permettant de désactiver l'option calcHygroThermal

Script Dymola: De même, un script Modelica peut être écrit pour automatiser la simulation des fichiers Modelica qui viennent d'être créés.

Remarque Dans les deux scripts précédents il faut être vigilent au chemin d'accès de chaque fichier. Suivant s'ils sont utiles et s'ils ont pu être simulés, les fichiers sont classés dans deux répertoires distincts `no_need_to_validate` et `validated`.

Résultats: Tous les fichiers à l'exception de `EnergyLabB665-with-materials.ifc` ont pu être simulés avec Dymola. Le fichier problématique renvoie de nombreuses erreurs lors de la simulation. Lors de la génération du fichier Modelica, un message indique qu'un mur est en collision avec un espace et n'est donc pas traité. On remarque effectivement qu'il manque un mur dans le fichier Modelica.

6.2 Outil de diagnostic

Des outils peuvent être créés afin de faciliter la détection des erreurs dans la conversion.

Analyse "manuelle" de fichiers IFC: Dans le cas du fichier `EnergyLabB665-with-materials.ifc`, l'identifiant IFC du mur non traité est affiché dans le message d'avertissement. Dans le fichier Modelica, aucun mur n'a cet identifiant. En ouvrant le fichier IFC avec Solibri on peut visualiser le mur en question et ainsi repérer à quel espace il est rattaché. On peut aussi facilement accéder à l'identifiant de l'espace et ainsi retrouver l'espace dans le fichier Modelica. On s'aperçoit alors que le nombre de murs liés à cet espace est erroné.

Analyse de fichiers IFC: L'outil AnalysisTool (`DataAnalysis/ifc_tools/IfcAnalysis/analysis_tool.py`) a été créé dans le but de faciliter le maniement du contenu des fichiers IFC.
Cet outil permet d'analyser un fichier IFC par type (IfcBuilding, IfcZone, IfcSpace, IfcSlab, IfcWall, IfcDoor, IfcWindow) et d'enregistrer les informations importantes et accessibles dans un dictionnaire.
Les informations récupérées (lorsqu'elles sont disponibles) sont les coordonnées de l'objet, les propriétés stockées dans BaseQuantities, les matériaux et le lien avec d'autres objets.

```
def analyseElement(self, element_type):
    elements = self.ifc_file.by_type(element_type)
    for element in elements:
        # Coordinates
        if element_type != "IfcZone":
            local_coordinates = ifcopenshell.util.placement.get_local_placement(element
    .ObjectPlacement)[0:3,3]
        # Quantities
        quantities = ifcopenshell.util.element.get_psets(element)
        base_quantities = {}
        try:
            base_quantities = quantities['BaseQuantities']
            bq_indicator = True
        except:
            print(element.GlobalId + " does not have BaseQuantities")
            bq_indicator = False
        # Materials
        if element_type != "IfcBuilding" and element_type != "IfcZone" and element_type
    != "IfcSpace":
            materials = self.findMaterials(element)
        # Store informations according to type
        ...
        if element_type == "IfcWall":
            self.analyseWall(element, local_coordinates, bq_indicator, base_quantities,
    materials)
        ...
```

Figure 19: Méthode analyseElement commune à tous les types d'éléments

```
def analyseWall(self, element, local_coordinates, bq_indicator, base_quantities,
    materials):
    if bq_indicator:
        self.walls[element.GlobalId] = [local_coordinates,
                                        (base_quantities['Length'], base_quantities['
    Width'], base_quantities['Height']),
                                        base_quantities['GrossFootprintArea'],
                                        base_quantities['GrossVolume'],
                                        materials,
                                        False # Set to true when saved in file
                                        ]
    else:
        self.walls[element.GlobalId] = [local_coordinates,
                                        "",
                                        "",
                                        "",
                                        materials,
                                        False # Set to true when saved in file
                                        ]
```

Figure 20: Méthode analyseWall permettant d'enregistrer les informations relatives à un mur

Diagnostic de fichiers IFC: À l'aide de l'outil AnalysisTool et du module Xlsxwriter, il est possible de créer un outil de diagnostic rapide. Les résultats de l'analyse avec AnalysisTool sous forme de dictionnaires peuvent être écrits dans un tableau, les rendant facilement lisibles.
Pour chaque fichier IFC analysé, un fichier xlsx est créé. Ce fichier contient une page par type d'élément présent (IfcBuilding, IfcZone, IfcSpace, IfcSlab, IfcWall, IfcDoor, IfcWindow).

Par exemple, grâce aux résultats présentés en annexe (voir section 8, page 25), il est facile de voir que le fichier `ACJasmin.ifc` contient un seul IfcBuilding, aucun IfcZone (pas de feuille IfcZone et colone Zone grisée), dix IfcSpace et cinq IfcSlab.

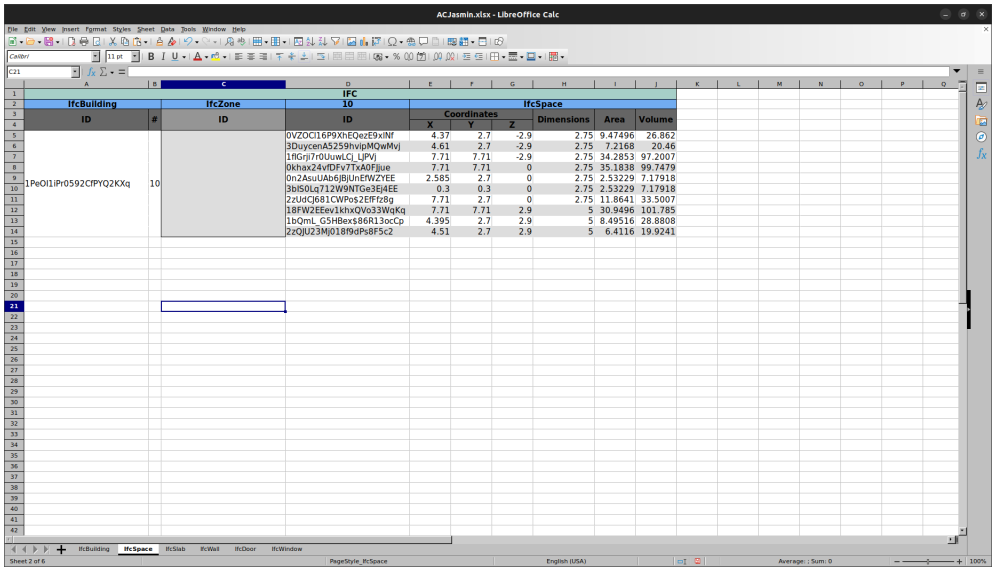

Figure 21: Extrait du résultat du diagnostic du fichier `ACJasmin.ifc`

De même, on constate que le bâtiment dans le fichier `TwoZones.ifc` contient deux IfcZones. La première zone contient un IfcSpace et la seconde en contient deux. Les six murs présents dans le fichier ne sont associés à aucun espace.

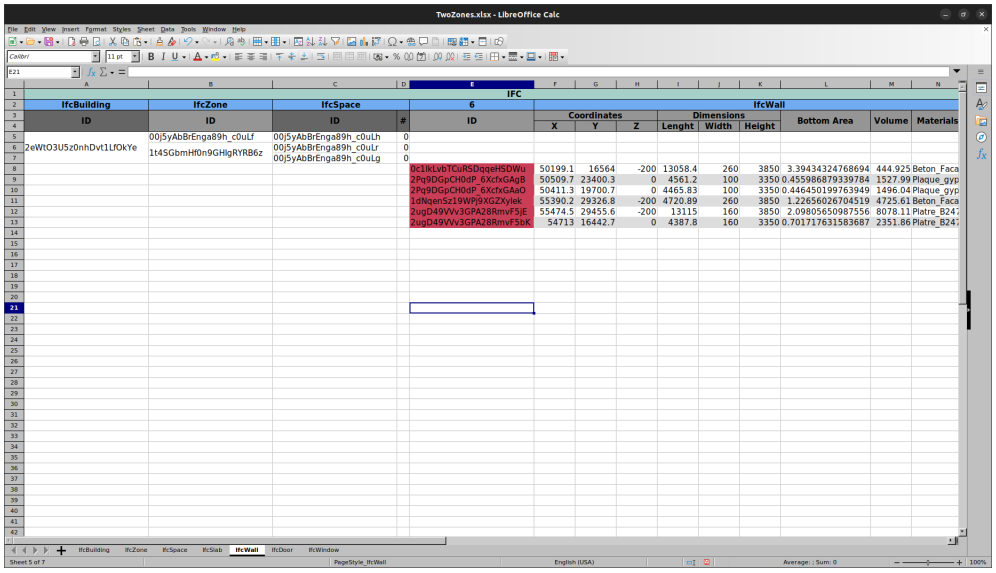

Figure 22: Extrait du résultat du diagnostic du fichier `TwoZones.ifc`

Comparaison de fichiers IFC: Il peut aussi être utile de comparer deux fichiers IFC, notamment deux fichiers provenant du même projet Revit ayant subit de légères modifications: options d'exportation différentes, modification des zones et espaces, ajout des propriétés des matériaux.
Cette comparaison permet alors de repérer quelles options d'exportation sont nécessaires ou si l'ajout des matériaux ne modifie pas d'autres propriétés.
Dans l'exemple suivant, les fichiers `OneZone.ifc` et `ThreeZones.ifc` sont issus du même projet Revit mais exportés avec différentes zones.

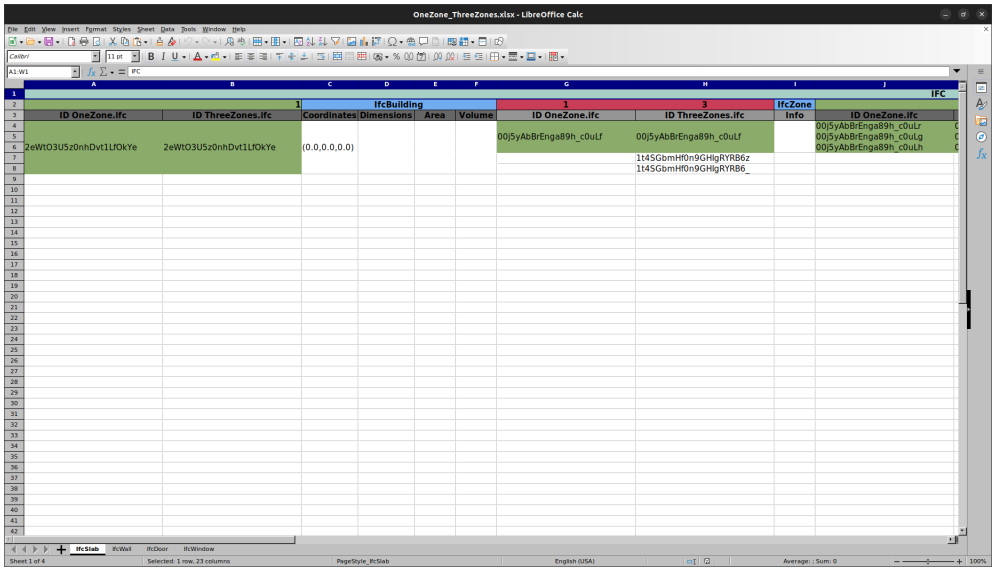

Figure 23: Résultat de la comparaison entre les fichiers `OneZone.ifc` et `ThreeZones.ifc`

Remarque || La gestion des pages et la mise en page peut être modifiée pour offrir plus de lisibilité suivant les besoins.

Comparaison de fichiers IFC et Modelica: Un outil pour comparer le contenu d'un fichier IFC et le contenu d'un fichier Modelica permet de déceler les erreurs lors de la conversion en fichier Modelica.

7 Étude et amélioration de l'outil de génération automatique de modèles

Grâce aux outils d'analyse et de comparaison, il est possible de déceler les erreurs lors de la conversion. Ces outils et les pytests automatiques permettront de définir si une modification effectuée dans BIM2Modelica est une dégradation ou une amélioration.

7.1 Fonctionnement

Le fonctionnement du générateur `IFC_MultiZoneBuildings_Modelica` utilisé dans BIM2Modelica utilise la fonction mapIFCtoBuildingDataModel du fichier `BIM2Modelica/CoTeTo_Generators/IFC_MultiZoneBuildings_Modelica/Filters/Filter01.py` pour récupérer toutes les informations importantes dans le fichier IFC et les stocker sous la forme d'objets définis dans `BIM2Modelica/CoTeTo_Generators/IFC_MultiZoneBuildings_Modelica/Filters/IfcLib/DataClasses.py`. Ces informations sont ensuite regroupées grâce à la fonction getGeneratorData qui renvoie un dictionnaire ayant la structure suivante, où les valeurs sont des listes d'objets:

```
{'materials':materials,
 'constructions':constructions,
 'zones':zones,
 'elementsOpaque':elementsOpaque,
 'elementsTransparent':elementsTransparent,
 'elementsDoor':elementsDoor,
 'conEleZon':conEleZon,
 'conEleAmb':conEleAmb,
 'conEleSol':conEleSol,
 'buildingSystem':buildingSystem}
```

Figure 24: Dictionnaire contenant les informations à écrire dans le fichier Modelica

7.2 Blacklist

Par défaut, CoTeTo/BIM2Modelica détecte les collisions entre objets. En effet, l'étude des fichiers `Filter01.py` et `Ifc2x3Lib.py` révèle l'existence d'une *blacklist*. Lors de la collecte des informations sur le bâtiment, la géométrie des éléments est analysée et les collisions sont détectées à l'aide d'outils Python-OCC. Par exemple, les IfcSlab et les IfcWall ayant plus de 90% de leur volume en commun avec un espace sont considérés comme des murs intérieurs négligeables pour la simulation thermique et ceux ayant plus de 1% de leur volume en collision sont considérés comme mal définis.

```
for w in all_walls:
    if w.GlobalId in WallInfo.keys():
        wall_shape = ifcopenshell.geom.create_shape(settings, w).geometry
        OCC.Core.BRepGProp.brepgprop_VolumeProperties(wall_shape, props)
        wall_volume = props.Mass()
        cmn_shape, area, BuilderCanWork = commonFace_andArea(wall_shape, space_volume)
        OCC.Core.BRepGProp.brepgprop_VolumeProperties(cmn_shape, props)
        if props.Mass() > wall_volume * 0.9:
            black_list_walls.append(w.GlobalId)
        elif props.Mass() > 0.01:
            ToBeRepairWall.append(w.GlobalId)
            black_list_walls.append(w.GlobalId)
```

Figure 25: Extrait de la fonction RelatedElementsWalls dans `Ifc2x3Lib`

Remarque || Dans la version du générateur utilisée, la liste ToBeRepairWall n'est pas réutilisée par la suite. C'est à dire que les murs en collision avec un espace ne sont pas corrigés.

Dans le cas du fichier `EnergyLabB665-with-materials.ifc` vu précédemment, un message d'avertissement indique qu'un mur est en collision avec un espace et qu'il est ignoré pour le reste du processus. Ce qui signifie que le fichier Modelica créé ne contient pas la véritable structure du bâtiment et donnerait lieu à des résultats de simulation faux s'il pouvait être simulé.

8 Conclusion

La simulation thermique de bâtiment s'inscrit dans les initiatives concrètes et actuelles pour lutter contre le changement climatique en favorisant le respect de l'environnement. Les pertes énergétiques peuvent être décelées et évaluées avant la construction ou dans l'objectif d'entreprendre la rénovation d'un bâtiment.

Les capacités (notamment matérielles) de calcul actuelles permettent de réaliser ce genre de simulation. Cependant, il est difficile de mettre au point un programme suffisamment flexible et robuste permettant de simuler des architectures variées, réalisées avec de nombreux matériaux et concues avec des logiciels différents.

De même que des travaux de recherche concernant les calculs des divers paramètres thermiques et météorologiques au cœur de la simulation sont nécessaires, des travaux de recherche sont indispensables afin d'utiliser pleinement les données matérielles d'un bâtiment. Cela consiste à extraire correctement l'ensemble des données géométriques, vérifier ou ajouter les informations concernant les paramètres thermiques (zones thermiques et propriétés des matériaux) et créer un fichier utilisable pour la simulation sans perdre d'information.

Ce travail requiert, en plus d'une bonne compréhension du problème, des compétences techniques dans différents domaines. Il est impératif de faire le lien entre les objets visualisés dans les logiciels de CAO et les données codées dans les fichiers grâce à une bonne maîtrise des logiciels et des langages de programmations utilisés. Il faut également faire preuve de patiente et d'une grande rigueur pour comprendre tous les enjeux, appréhender les difficultés rencontrées et trouver les solutions possibles. Le tout doit être réalisé en conservant un espace de travail partagé organisé et compréhensible pour tous et en communicant les résultats à l'ensemble des membres de l'équipe de recherche.

Bibliographie

[1] Anaconda. https://www.anaconda.com/products/distribution. Accessed: 2022-07-15.

[2] Buildingsystems. https://modelica-buildingsystems.de/. Accessed: 2022-07-15.

[3] Cemosis. http://www.cemosis.fr/. Accessed: 2022-07-15.

[4] Cemosis. https://sites.google.com/a/cemosis.fr/www/qui-sommes-nous. Accessed: 2022-07-15.

[5] Cemosis. https://www.linkedin.com/company/cemosis/?originalSubdomain=fr. Accessed: 2022-07-15.

[6] Cemosis. http://www.cemosis.fr/projects/. Accessed: 2022-07-15.

[7] Code templating tool (coteto). https://github.com/UdK-VPT/CoTeTo. Accessed: 2022-07-15.

[8] Dymola. https://www.3ds.com/fr/produits-et-services/catia/produits/dymola/. Accessed: 2022-07-15.

[9] Github. https://github.com/. Accessed: 2022-07-15.

[10] Ifcopenshell. http://ifcopenshell.org/. Accessed: 2022-07-15.

[11] Modelica. https://modelica.org/. Accessed: 2022-07-15.

[12] Pyhton-occ. https://dev.opencascade.org/project/pythonocc. Accessed: 2022-07-15.

[13] Pytest. https://docs.pytest.org/en/7.1.x/#. Accessed: 2022-07-15.

[14] Python. https://www.python.org/. Accessed: 2022-07-15.

[15] Revit. https://www.autodesk.fr/products/revit. Accessed: 2022-07-15.

[16] Slack. https://slack.com. Accessed: 2022-07-15.

[17] Solibri. https://www.solibri.com/. Accessed: 2022-07-15.

[18] Téléchargement anaconda. https://www.anaconda.com/products/distribution. Accessed: 2022-07-15.

[19] Téléchargement solibri. https://www.solibri.com/download-solibri-anywhere?step=1. Accessed: 2022-07-15.

[20] Winehq. https://www.winehq.org/. Accessed: 2022-07-15.

[21] Xlsxwriter. https://github.com/jmcnamara/XlsxWriter. Accessed: 2022-07-15.

[22] Zoom. https://explore.zoom.us. Accessed: 2022-07-15.

[23] Énergie dans les bâtiments. https://www.ecologie.gouv.fr/energie-dans-batiments. Accessed: 2022-07-15.

[24] Javier Cladellas. Indoor and outdoor data processing and analysis for building energy and comfort assessement application. https://github.com/feelpp/ktirio.dashboard/blob/diagnostic_tools/rapport/rapport.pdf. Accessed: 2022-07-15.

[25] Ministère de l'enseignement supérieur et de la recherche. Les cifre. https://www.enseignementsup-recherche.gouv.fr/fr/les-cifre-46510. Accessed: 2022-07-15.

[26] Feel++. Aerothermal simulation in a building. https://www.youtube.com/watch?v=DWf3OKYT3WE&ab_channel=Feel%2B%2B. Accessed: 2022-07-15.

Annexes

Installation des outils

Anaconda: Sous Windows, un fichier exécutable est disponible sur la page de téléchargement[18].
Sous Linux, un fichier shell peut être téléchargé et exécuté dans un terminal sur la page de téléchargement[18].

IfcOpenShell et Python-OCC: La version 0.6.0b0 d'IfcOpenShell et la version 7.5.1 de Python-OCC ont été installées sur un environnement Conda reposant sur la version 3.8.13 de Python en utilisant les commandes:

```
conda install -c conda-forge ifcopenshell
conda install -c conda-forge/label/cf202003 ifcopenshell

conda install -c conda-forge pythonocc-core
conda install -c conda-forge/label/cf202003 pythonocc-core
```

CoTeTo: L'outil Code Templating Tool peut être téléchargé directement depuis le dépôt GitHub[7]. Il existe trois façons d'utiliser CoTeTo: à l'aide de l'interface graphique, en ligne de commande ou avec un script Python.

Interface graphique: Il faut lancer le script `BIM2Modelica/CoTeTo/scripts/CoTeTo-gui.py`. L'interface permet de choisir le générateur, le fichier IFC à convertir et le chemin d'accès au fichier généré. On peut suivre l'évolution du processus et lire les éventuels messages d'erreur dans l'onglet Messages.

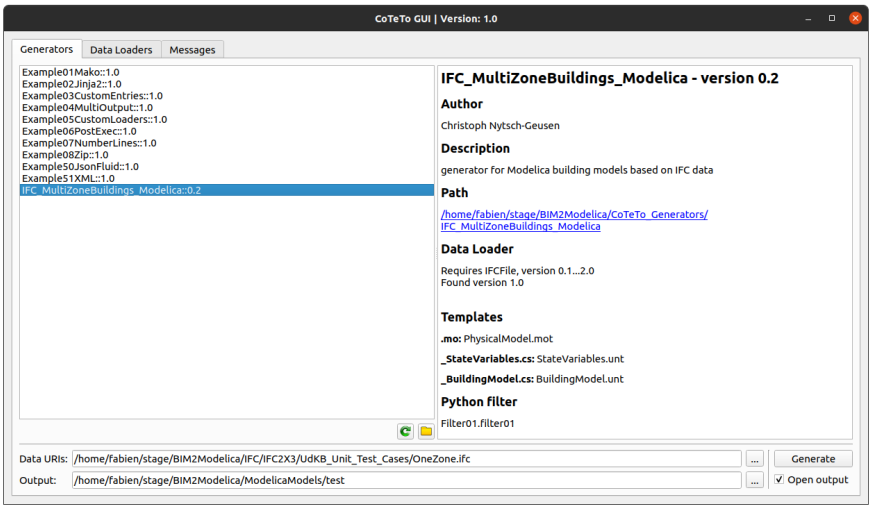

Figure 26: Interface graphique de CoTeTo

Lignes de commande: Il suffit d'utiliser les commandes suivantes pour utiliser CoTeTo dans un terminal. La première commande affiche l'aide, la deuxième permet de générer un fichier Modelica à partir d'un fichier IFC et la dernière commande permet d'effectuer une comparaison grossière entre le résultat attendu et le résultat obtenu (en ignorant les 277 premiers octets, on évite la ligne dans l'entête du fichier contenant l'heure et la date de création du fichier, cependant la première ligne, affichant les paquets utilés, n'est pas vérifiée).

```
python CoTeTo-cli.py --help
python CoTeTo-cli.py -p ../Generators -g ? -o [input_file] [output_file] A VERIFIER
cmp -l -b -i 277 [file1] [file2]
```

Il faut cependant être vigilant et modifier le fichier `CoTeTo/CoTeTo/Generators/IFC_MultiZoneBuilding_Modelica/Package.inf` pour obtenir le bon entête et pied de page (adapter ModelicaLibPath et ModelicaMosPath).

Scripts Python: CoTeTo est aussi utilisable en tant que module Python. Voici un exemple de script utilisé pour générer les fichiers avec CoTeTo.

```
#!/usr/bin/env python3
import sys
import os
os.mkdir('./generatedCode')
os.chdir('./generatedCode')
sys.path.append("../../../../CoTeTo")
from CoTeTo.Controller import Controller
# Initialise CoTeTo controller
con = Controller()
# Initialise CoTeTo generator
gen = con.generators['IFC_MultiZoneBuildings_Modelica::0.2']
# Lists of files
files = [('OneZone.ifc','OneZone'),('TwoZones.ifc','TwoZones')]
# Generate files
for f in files:
    gen.execute(('../'+f[0], ), f[1])
```

Là aussi il faut être vigilant et modifier le fichier `CoTeTo/CoTeTo/Generators/IFC_MultiZoneBuilding_Modelica/Package.inf` pour obtenir le bon entête et pied de page (adapter ModelicaLibPath et ModelicaMosPath). Une solution à ce problème est proposée dans le script pour générer les fichiers de référence.

Dymola: La procédure d'installation du logiel se fait en exécutant les lignes de commandes suivantes:

```
ln /usr/lib/x86_64-linux-gnu/libevent-2.1.so.7 /usr/lib/x86_64-linux-gnu/libevent-2.0.
    so.5
wget -O Dymola.zip https://www.dropbox.com/s/pic16nfu9d9vy63/Dymola_2021.AM_CAT_Dymola.
    Linux64.1-1.zip?dl=0 &&\
unzip -j Dymola.zip AM_CAT_Dymola.Linux64/1/linux_x86_64/dymola-2021.1-1.x86_64.rpm &&\
alien -k dymola-2021.1-1.x86_64.rpm &&\
apt install ./dymola_2021.1-1_amd64.deb &&\
rm dymola-2021.1-1.x86_64.rpm dymola_2021.1-1_amd64.deb Dymola.zip

dymola-2021-x86_64
```

Il faut ensuite activer la licence. Pour cela, naviguer dans les menus tools, licence, set-up, set-up. Dans le cadre server name saisir: gaya.math.unistra.fr
La procédure pour lancer une simulation est la suivante:

- Charger la librairie BuildingSystems (version master?) en navigant dans le menu File ➔Open ➔Load. Naviguer dans `BuildingSystmes/BuildingSystems` et charger le fichier `package.mo`.
- Ouvrir le fichier Modelica à simuler en navigant dans le menu File ➔Open ➔Open.
- Lancer la simulation depuis l'onglet Simulation ➔Simulate.

Remarque || Le menu File ➔Working Directory permet de définir l'emplacement où sont stockés les fichiers de simulation.

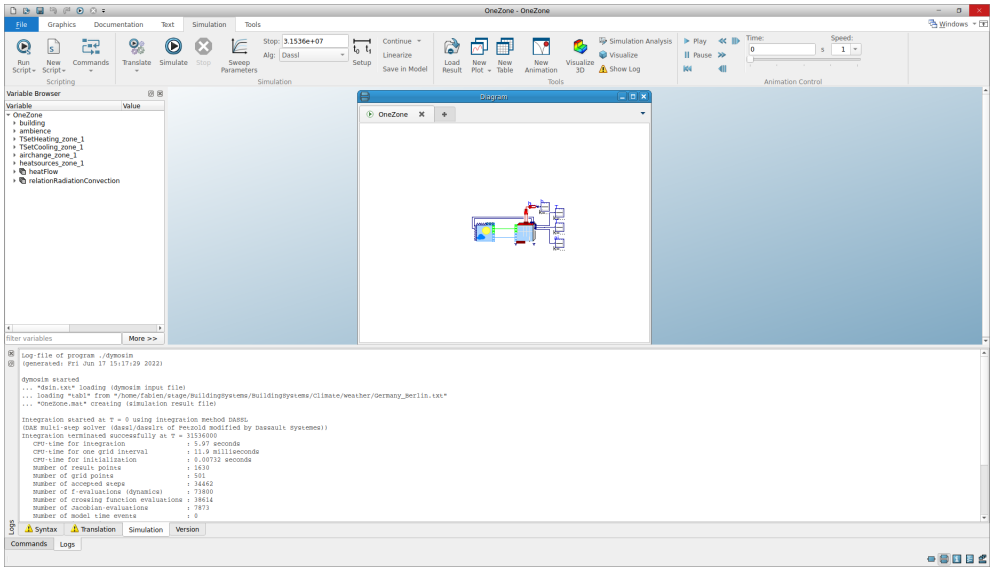

Figure 27: Interface grahique du logiciel Dymola

Solibri: Sous Windows, le fichier exécutable pour installer Solibri est accessible après avoir suivi la procédure d'enregistrement[19]. Pour utiliser Solibri dans une distribution Linux, il faut utiliser WineHQ[20].

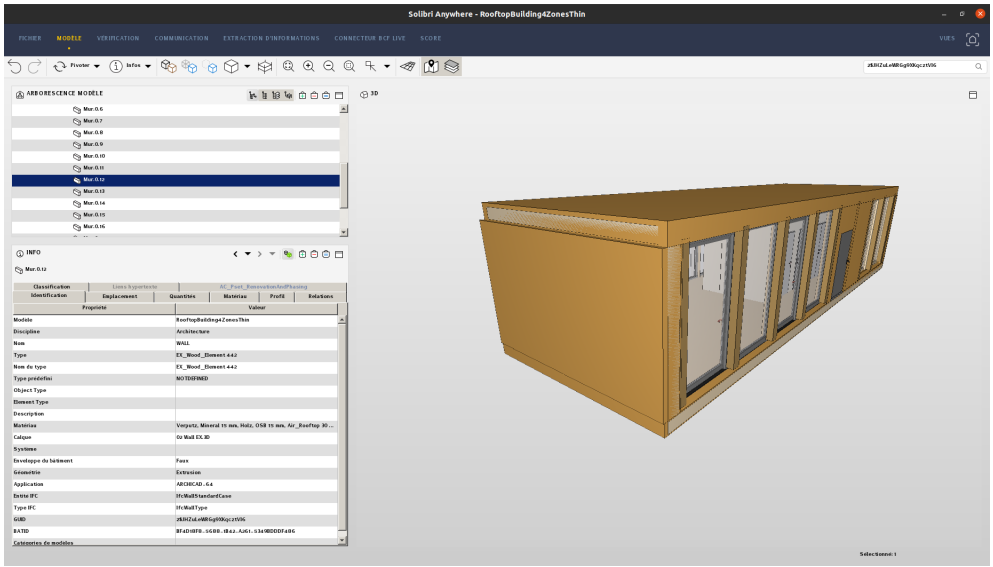

Figure 28: Interface grahique du logiciel Solibri

Résultats des diagnostics

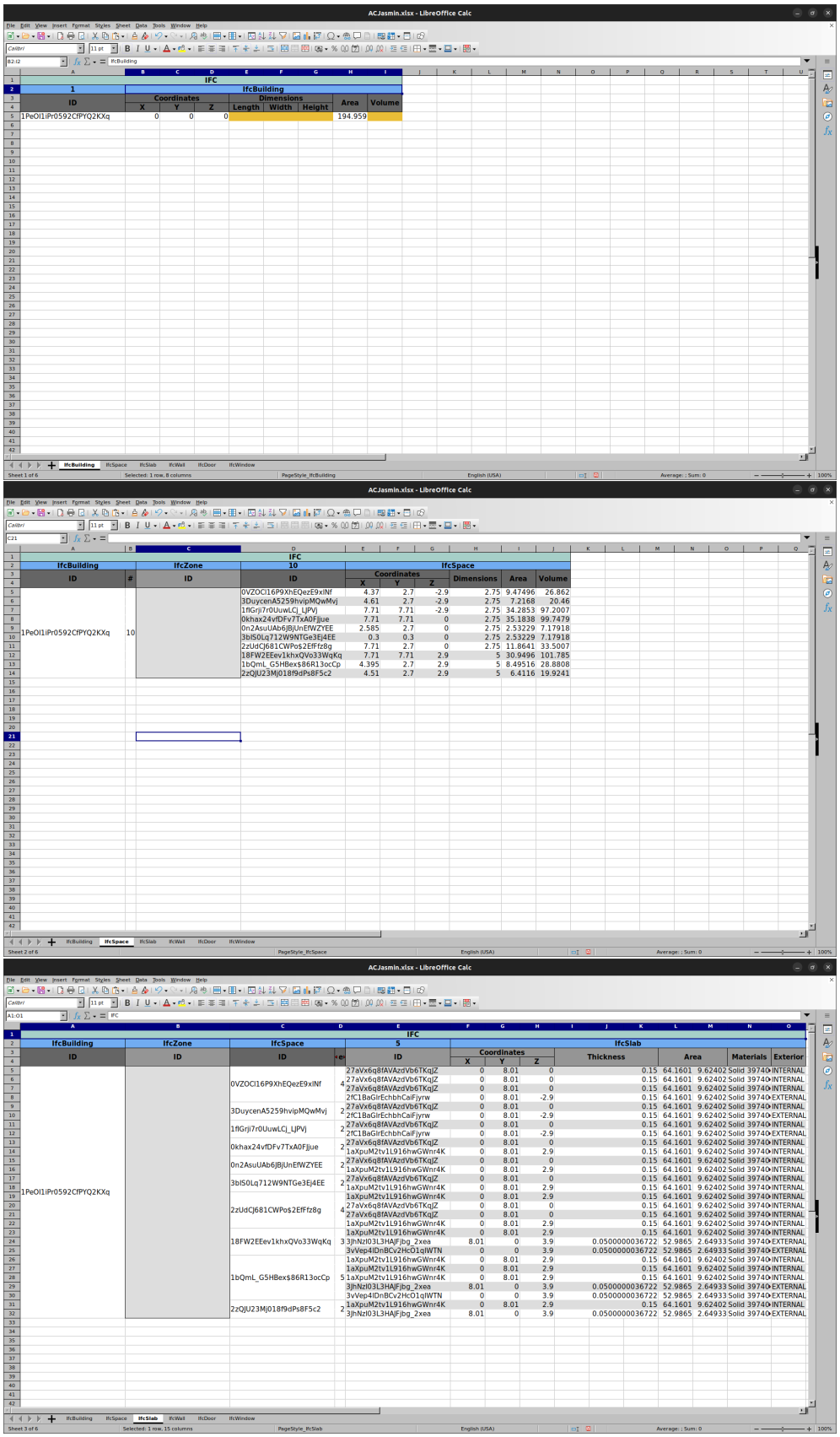

Figure 29: Résultat du diagnostic du fichier `ACJasmin.ifc`

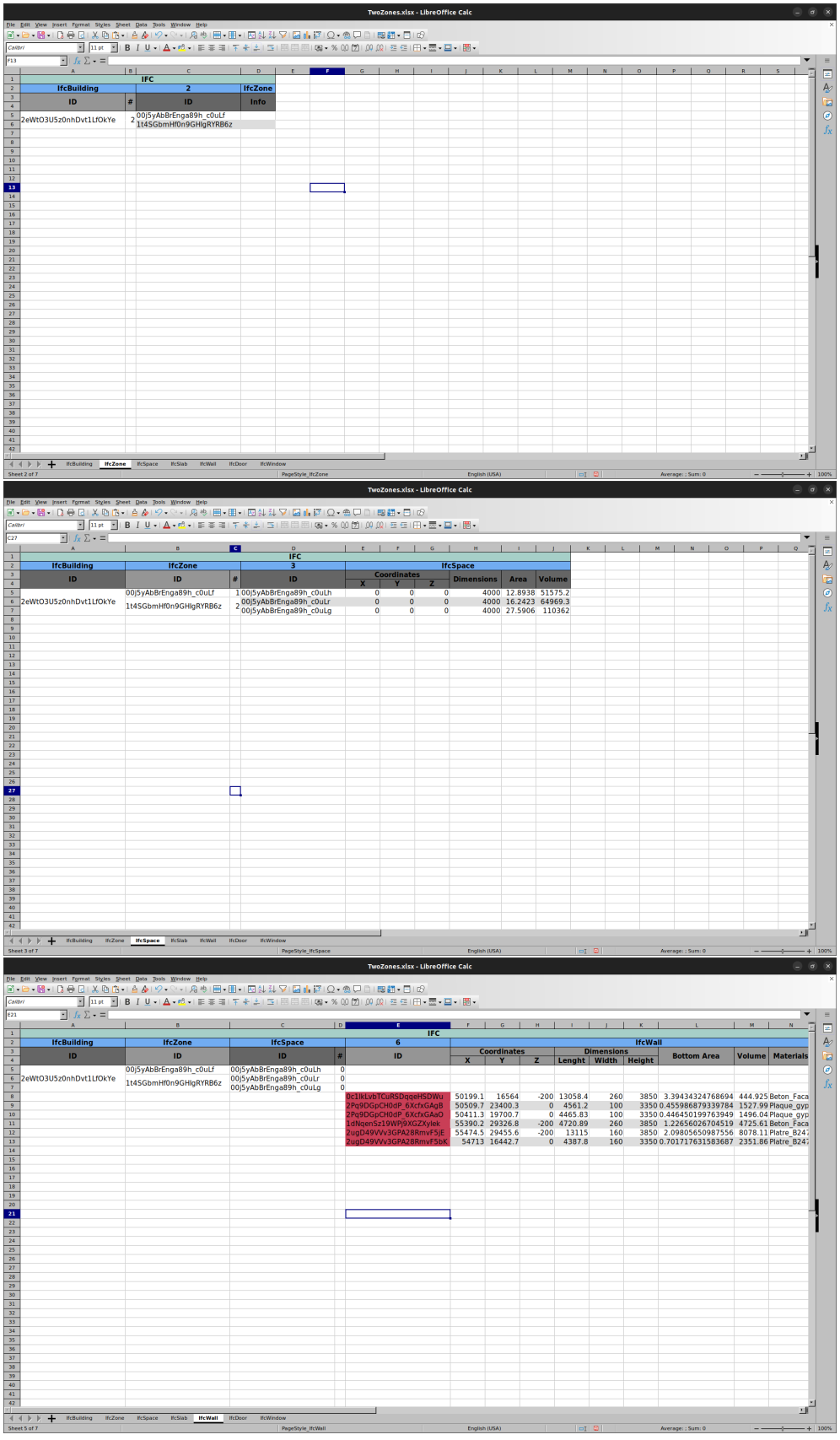

Figure 30: Résultat du diagnostic du fichier `TwoZones.ifc`