

Projet de session :

**Livrable no 2 - Modèle de conception et Architecture logique
Génie logiciel orienté objet GLO-2004**

HeatMyFloor
Travail présenté au :

Département d'informatique et de génie logiciel
Université Laval
Automne 2025

Sayi Camara sayi.camara.1@ulaval.ca
Staëlle Eugène staelle.eugene.1@ulaval.ca
Jacky Masson Kemajou de Tonga jacky-masson.kemajou-de-tonga.1@ulaval.ca
Hadadak Ndwedeme Dilane hedadak.ndwedeme-dilane.1@ulaval.ca
Charles-Auguste Plouffe charles-auguste.plouffe.1@ulaval.ca

Table of Contents

1	INTRODUCTION	3
2	DIAGRAMME DE CLASSES DE CONCEPTION	4
CETTE COUCHE REPRÉSENTE LE COEUR DE LA LOGIQUE MÉTIER DE L'APPLICATION, AVEC LES ENTITÉS ET LEURS COMPORTEMENTS.....	6	
<i>Contraintes</i>	6	
<i>Pièce</i>	6	
<i>Meuble</i>	6	
<i>La classe abstraite Meuble représente un meuble générique. Elle possède une forme (Polygon), un ID (UUID) et un indicateur estSelectionné (bool)</i>	6	
<i>MainWindow</i>	8	
<i>DrawingPanel (Afficheur)</i>	8	
<i>MainDrawer</i>	8	
<i>HeatMyFloorController</i>	9	
3	ARCHITECTURE LOGIQUE	12
3.1	DIAGRAMME DE PACKAGES	12
4 🔗	<i>Relations entre packages</i>	13
4	DIAGRAMMES DE SÉQUENCE DE CONCEPTION (DSC)	14
4.1	DÉTERMINATION DE L'ÉLÉMENT SUR LEQUEL A LIEU UN CLIC DE SOURIS	14
4.1.1	<i>Déterminer les coordonnées (x, y) en pouces lors d'un clic de souris</i>	14
4.1.2	<i>Déterminer l'élément sur lequel a lieu le clic</i>	15
4.2	DSC AJOUT D'UNE DOUCHE	17
4.3	DSC AFFICHAGE DE LA VUE	18
5	PLAN DE TRAVAIL MIS À JOUR (GANTT)	19
6	CONTRIBUTION DES MEMBRES DE L'ÉQUIPE	20

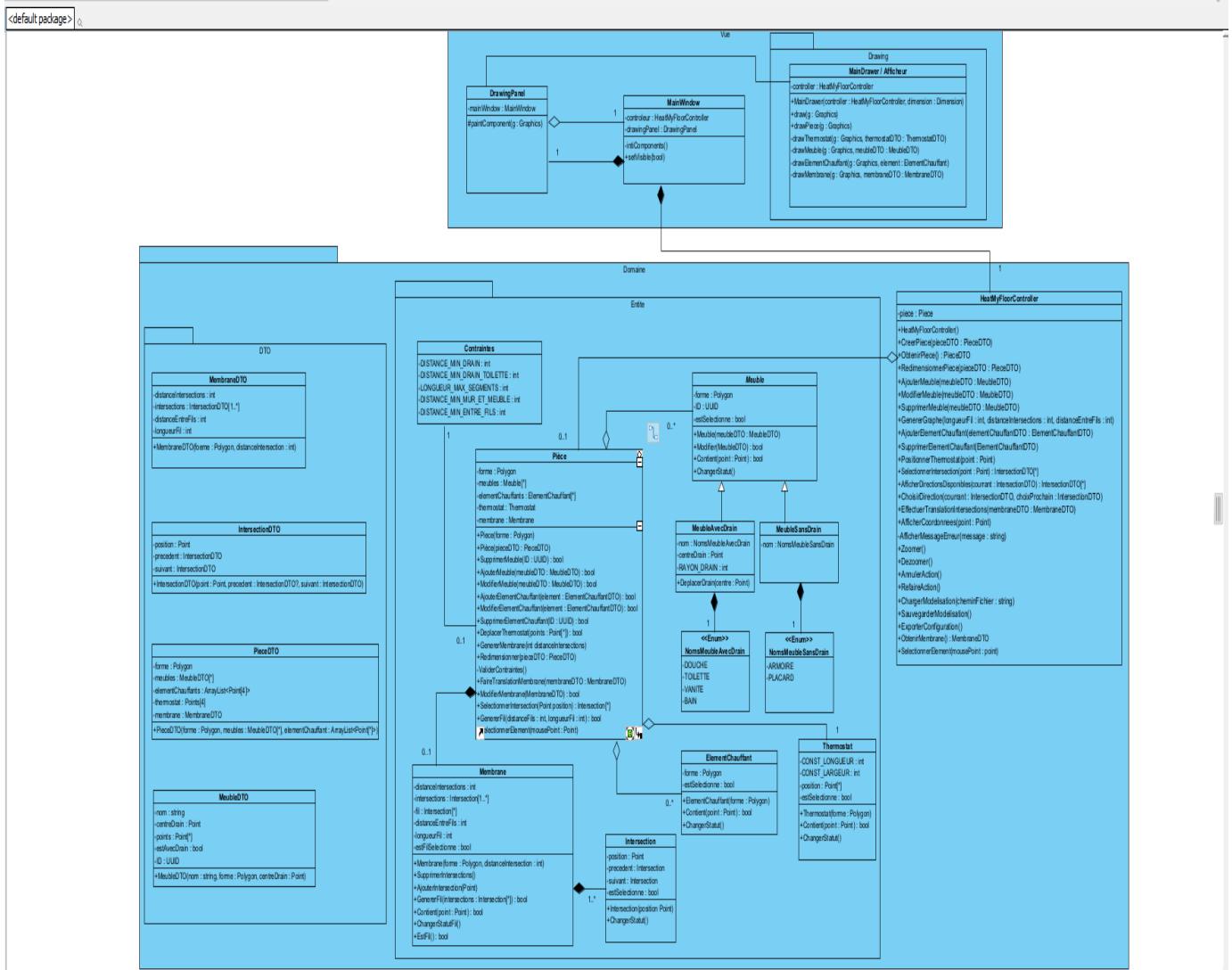
1 Introduction

Ce rapport présente le modèle de conception et l'architecture logique de l'application **HeatMyFloor**, qui permet de planifier l'installation d'un plancher chauffant dans une pièce. L'objectif est de fournir aux utilisateurs, et générer automatiquement une trace optimale pour le fil chauffant tout en respectant les contraintes de sécurité.

Notre conception s'articule autour d'une architecture en couches séparant la présentation, le contrôle et le domaine. Nous avons appliqué le principe du contrôleur de Larman pour centraliser la coordination entre l'interface utilisateur et la logique métier. Une classe **Afficheur** dédiée gère le rendu graphique dans JPanel avec Swing, incluant les transformations de coordonnées et la gestion du zoom autour du curseur de la souris.

Le rapport présente le diagramme de classes de conception avec les descriptions des classes et relations, l'architecture logique en packages, trois diagrammes de séquence de conception (détermination de l'élément clique, ajout d'une douche, affichage de la vue), le plan de travail révisé et les contributions des membres de l'équipe. Ce travail de conception a nécessité plusieurs itérations et un prototypage préliminaire pour valider les aspects techniques critiques.

2 Diagramme de classes de conception



2.1.1.1 Description Générale

Ce diagramme de classes de conception met l'accent sur les classes, leurs attributs, leurs méthodes et les relations qui les unissent. L'architecture est structurée autour des principes de séparation des préoccupations, notamment en distinguant les couches de présentation (Vue), de logique métier (Domaine/Entité) et de transfert de données (DTO), ainsi qu'en appliquant le principe du contrôleur de Larman.

2.1.1.2 Architecture Générale

Le diagramme de classes est organisé en plusieurs sections logiques :

- **DTO (Data Transfer Objects)** : Ces classes sont utilisées pour transférer des données entre les différentes couches de l'application, notamment entre la couche de présentation et la couche de logique métier. Elles sont des représentations simplifiées des entités du domaine.
- **Domaine/Entité** : Cette section contient les classes qui représentent la logique métier et les données persistantes de l'application. C'est le cœur du système, où les règles métier sont implémentées.
- **Vue** : Cette couche est responsable de l'affichage de l'interface utilisateur et de la gestion des interactions avec l'utilisateur. Elle est composée de classes graphiques.
- **Contrôleur** : La classe **HeatMyFloorController** agit comme le contrôleur principal, orchestrant les interactions entre la Vue et le Domaine, conformément au principe du contrôleur de Larman.

2.1.1.3 Couche DTO (Data Transfer Objects)

Cette couche regroupe les objets de transfert de données, qui sont des représentations légères des entités du domaine, utilisées pour la communication.

MembraneDTO

La classe **MembraneDTO** est un objet de transfert de données pour la **Membrane**. Elle contient des informations telles que `distanceIntersections` (int), `intersections` (`ArrayList<IntersectionDTO>`), `distanceEntreFils` (int) et `longueurFil`(int). Son constructeur prend une forme (Polygone) et une `distanceIntersection` (int).

IntersectionDTO

La classe **IntersectionDTO** représente une intersection dans le contexte des DTO. Elle possède une position (Point), un précédent (IntersectionDTO) et un suivant (IntersectionDTO), permettant de définir des chaînes d'intersections. Son constructeur permet d'initialiser ces attributs.

PieceDTO

La classe **PieceDTO** est DTO pour une **Pièce**. Elle encapsule la forme (Polygon) de la pièce, une liste de meubles (`ArrayList<MeubleDTO>`), une liste d'`elementChauffants` (`ArrayList<Point [4]>`), représentant les quatre coins de chaque élément chauffant, un thermostat (Point [4]) et une membrane (MembraneDTO). Son constructeur permet de créer une instance avec ces informations.

MeubleDTO

La classe **MeubleDTO** est le DTO pour un **Meuble**. Elle contient le nom (string), le centreDrain (Point), les points (Point []), un indicateur estAvecDrain (bool) et un ID (UUID). Son constructeur permet de l'initialiser avec un nom, une forme et un centre de drain.

2.1.1.4 Couche Domaine / Entité

Cette couche représente le cœur de la logique métier de l'application, avec les entités et leurs comportements.

Contraintes

La classe **Contraintes** est une classe utilitaire ou un ensemble de constantes définissant des valeurs minimales ou maximales pour diverses mesures, telles que DISTANCE_MIN_DRAIN, DISTANCE_MIN_DRAIN_TOILETTE, LONGUEUR_MAX_SEGMENTS, DISTANCE_MIN_MUR_ET_MEUBLE, et DISTANCE_MIN_ENTRE_FILS. Ces valeurs sont de type int.

Pièce

La classe **Pièce** est une entité centrale du domaine. Elle est composée d'une forme (Polygon), d'une collection de meubles (ArrayList<Meuble>), d'une collection d'elementChauffants (ArrayList<ElementChauffant>), d'un thermostat (Thermostat) et d'une membrane (Membrane).

Elle offre des méthodes pour manipuler ces composants, comme **AjouterMeuble**, **SupprimerMeuble**, **ModifierMeuble**, **AjouterElementChauffant**, **SupprimerElementChauffant**, **ModifierElementChauffant**, **DeplacerThermostat**, **GénererMembrane**, **Redimensionner**, **FaireTranslationMembrane**, **ModifierMembrane**, **SelectionnerIntersection**, **GénererFil**, et **SelectionnerElement**.

Elle possède deux constructeurs : un prenant une forme (Polygon) et un autre prenant un **PièceDTO** pour faciliter la création à partir de données transférées.

Meuble

La classe abstraite **Meuble** représente un meuble générique. Elle possède une forme (Polygon), un ID (UUID) et un indicateur estSelectionné (bool).

Elle définit un constructeur prenant un **MeubleDTO** pour créer une instance de **Meuble** à partir de données transférées, une méthode **Modifier** (probablement +Modifier (meubleDTO: MeubleDTO)) pour modifier ses propriétés, une méthode pour vérifier si un point est contenu (**Contient**), et une méthode **ChangerStatut** pour modifier l'état de sélection.

Elle est la classe parente de **MeubleAvecDrain** et **MeubleSansDrain**.

MeubleAvecDrain

La classe **MeubleAvecDrain** hérite de **Meuble** et représente un meuble spécifique qui possède un drain. Elle a un nom (de type NomsMeubleAvecDrain), un centreDrain (Point) et un RAYON_DRAIN (int). Elle inclut une méthode **DeplacerDrain** pour repositionner le drain.

MeubleSansDrain

La classe **MeubleSansDrain** hérite de **Meuble** et représente un meuble sans drain. Elle a un nom (de type NomsMeubleSansDrain).

NomsMeubleAvecDrain (Enum)

L'énumération **NomsMeubleAvecDrain** définit les types de meubles avec drain, tels que DOUCHE, TOILETTE, VANITE, BAIN.

NomsMeubleSansDrain (Enum)

L'énumération **NomsMeubleSansDrain** définit les types de meubles sans drain, tels que ARMOIRE, PLACARD.

Membrane

La classe **Membrane** représente la membrane chauffante. Elle contient distanceIntersections (int), intersections (ArrayList<Intersection>), fil (ArrayList<Intersection>), distanceEntreFils (int), longueurFil (int) et estFilSelectionne (bool).

Elle propose des méthodes pour **SupprimerIntersections**, **AjouterIntersection**, **GénererFil**, vérifier si un point est contenu (**Contient**), **ChangerStatutFil**, et **EstFil**. Son constructeur prend une forme (Polygon) et une distanceIntersection (int).

Intersection

La classe **Intersection** représente un point d'intersection dans la membrane. Elle a une position (Point), un précédent (Intersection) et un suivant (Intersection), ainsi qu'un indicateur estSelectionné (bool). Elle a un constructeur prenant une position et une méthode **ChangerStatut** pour modifier l'état de sélection.

ElementChauffant

La classe **ElementChauffant** représente un élément chauffant. Elle a une forme (Polygon) et un indicateur estSelectionné (bool). Elle possède un constructeur prenant une forme, et des méthodes **Contient** et **ChangerStatut**.

Thermostat

La classe **Thermostat** représente un thermostat. Elle définit des constantes CONST_LONGUEUR et CONST_LARGEUR (int), une liste de position (Point []) et un indicateur estSelectionné (bool). Elle a un constructeur prenant une forme (Polygon), et des méthodes **Contient** et **ChangerStatut**.

2.1.1.5 Couche Vue

Cette couche est responsable de l'affichage graphique de l'application.

MainWindow

La classe **MainWindow** représente la fenêtre principale de l'application. Elle contient une référence au contrôleur (HeatMyFloorController) et au drawingPanel (DrawingPanel). Elle gère l'initialisation des composants (initComponents()) et la visibilité de la fenêtre (setVisible(bool)).

DrawingPanel (Afficheur)

La classe **DrawingPanel** est le composant graphique principal où le dessin est effectué. Elle hérite de JPanel et utilise les classes de Swing, notamment **Graphics**, pour effectuer le rendu visuel. La méthode #paintComponent (g : Graphics) est responsable de l'affichage des éléments. C'est cette classe qui agit comme l'Afficheur demandé, en déléguant les opérations de dessin spécifiques à la classe **MainDrawer**.

MainDrawer

La classe **MainDrawer** est une classe utilitaire dédiée aux opérations de dessin. Elle prend en charge le contrôleur (HeatMyFloorController) et la dimension (Dimension) de la zone de dessin. Elle fournit des méthodes génériques draw(g : Graphics) ainsi que des méthodes spécifiques pour dessiner différents éléments du domaine : **drawPiece**, **drawThermostat**, **drawMeuble**, **drawElementChauffant**, et **drawMembrane**. Ces méthodes prennent un objet Graphics et le DTO correspondant à l'élément à dessiner, assurant ainsi une séparation claire entre la logique de dessin et les données.

2.1.1.6 Contrôleur (Principe du Contrôleur de Larman)

HeatMyFloorController

La classe **HeatMyFloorController** est le contrôleur principal de l'application, appliquant le principe du contrôleur de Larman. Elle agit comme un point d'entrée unique pour les événements du système, recevant les requêtes de la **MainWindow** (Vue) et coordonnant les actions avec les entités du Domaine. Elle ne contient pas de logique métier complexe elle-même, mais délègue les responsabilités aux objets du domaine appropriés.

Le **HeatMyFloorController** maintient une référence à une **Pièce** (-piece: Piece). Ses méthodes publiques correspondent aux opérations système ou aux actions utilisateur, telles

que **CreerPiece**, **ObtenirPiece**, **RedimensionnerPiece**, **AjouterMeuble**, **ModifierMeuble**, **SupprimerMeuble**, **GénererGraphe**, **AjouterElementChauffant**, **SupprimerElementChauffant**, **PositionnerThermostat**, **SelectionnerIntersection**, **AfficherDirectionsDisponibles**, **ChoisirDirection**, **EffectuerTranslationIntersections**, **AfficherCoordonnees**, **AfficherMessageErreur**, **Zoomer**, **Dezoomer**, **AnnulerAction**, **RefaireAction**, **ChargerModelisation**, **SauvegarderModelisation**, **ExporterConfiguration**, **ObtenirMembrane**, et **SelectionnerElement**.

Ces méthodes reçoivent des DTO en entrée et/ou retournent des DTO, facilitant la communication avec la couche de présentation sans exposer directement les objets du domaine. Par exemple, **CreerPiece(pieceDTO: PieceDTO)**, **ObtenirPiece() : PieceDTO**, et **AjouterMeuble(meubleDTO: MeubleDTO)** servent de mécanismes d'accès et de manipulation des données, remplissant ainsi le rôle des getters/setters traditionnels dans un contexte de contrôleur.

Les méthodes privées **-ValiderContraintes()** et **-AfficherMessageErreur(message : string)** suggèrent une gestion interne des contraintes et des messages d'erreur, qui sont des responsabilités typiques d'un contrôleur.

2.1.1.7 Relations entre les Classes

Les relations entre les classes sont cruciales pour comprendre la structure du système :

Composition et Agrégation dans le Domaine :

- Une **Pièce** contient plusieurs **Meuble** (agrégation, 0..),
des **ElementChauffant** (agrégation, 0..), un **Thermostat** (composition, 1) et
une **Membrane** (composition, 1). L'agrégation (losange vide) indique que
les **Meuble** et **ElementChauffant** peuvent exister indépendamment de la **Pièce**. La
composition (losange plein) pour le **Thermostat** et la **Membrane** indique que leur
durée de vie est liée à celle de la **Pièce** - ils n'existent pas sans elle.
- Une **Membrane** est composée d'une ou plusieurs **Intersection** (agrégation, 1..*). Le
fil de la **Membrane** est également une collection d'**Intersection**.
- Une **Intersection** peut avoir une **Intersection** précédent et
une **Intersection** suivant, formant une chaîne doublement liée.
- La classe abstraite **Meuble** est spécialisée
en **MeubleAvecDrain** et **MeubleSansDrain** (héritage).

Relations entre DTO et Entités :

- Les DTO (par exemple, **PieceDTO**, **MeubleDTO**, **MembraneDTO**, **IntersectionDTO**)
sont des représentations des entités correspondantes du domaine
(**Pièce**, **Meuble**, **Membrane**, **Intersection**). Les contrôleurs et les vues utilisent les
DTO pour échanger des données, tandis que la logique métier opère sur les entités.

Relations dans la Vue:

- La **MainWindow** contient une **DrawingPanel** (agrégation, 1) et interagit avec
le **HeatMyFloorController** (agrégation, 1).
- La **DrawingPanel** utilise la **MainDrawer** pour effectuer les opérations de dessin
(agrégation, 1).
- La **MainDrawer** a une référence au **HeatMyFloorController** (agrégation, 1) pour
potentiellement récupérer des données ou déclencher des actions.

Relations avec le Contrôleur :

- Le **HeatMyFloorController** maintient une référence à une **Pièce** (agrégation, 1), ce
qui est cohérent avec le principe du contrôleur de Larman où le contrôleur est
responsable de la coordination des opérations sur les objets du domaine.

2.1.1.8 Respect des Contraintes Spécifiées

Principe du Contrôleur de Larman

La classe **HeatMyFloorController** est clairement identifiée comme le contrôleur central. Elle gère les requêtes de la Vue et délègue les opérations aux objets du domaine (**Pièce** et ses composants), sans contenir de logique métier complexe elle-même. Elle utilise des DTO pour la communication avec la Vue, ce qui est une bonne pratique pour un contrôleur.

Classe « Afficheur »

La classe **DrawingPanel** remplit ce rôle. Elle hérite de JPanel et utilise les classes de Swing (**Graphics**) pour effectuer le rendu visuel. La **MainDrawer** est une classe auxiliaire qui centralise les méthodes de dessin spécifiques pour les différents éléments (**Pièce**, **Thermostat**, **Meuble**, **ElementChauffant**, **Membrane**), en prenant des DTO en entrée. Cette séparation permet une organisation claire du code de dessin.

Aggrégation ou Composition pour les Attributs

Le diagramme utilise des symboles d'agrégation (losange vide) et de composition (losange plein) pour représenter les relations où une classe contient une autre. L'agrégation est utilisée pour les **Meuble** et **ElementChauffant** qui peuvent exister indépendamment, tandis que la composition est utilisée pour le **Thermostat** et la **Membrane** dont la durée de vie est liée à la **Pièce**. Cela est conforme à la contrainte.

Get/Set

La contrainte stipule que les get/set ne sont pas obligatoires, sauf pour le contrôleur de Larman. Le diagramme ne les montre pas explicitement pour la plupart des classes, ce qui allège la lecture. Le **HeatMyFloorController** présente des méthodes publiques telles que **CreerPiece()**, **ObtenirPiece()**, **RedimensionnerPiece()**, **AjouterMeuble()**, etc., qui servent de mécanismes d'accès et de manipulation des données, remplissant ainsi le rôle des getters/setters traditionnels dans un contexte de contrôleur.

Appréciation Générale, Formalisme et Niveau de Détails

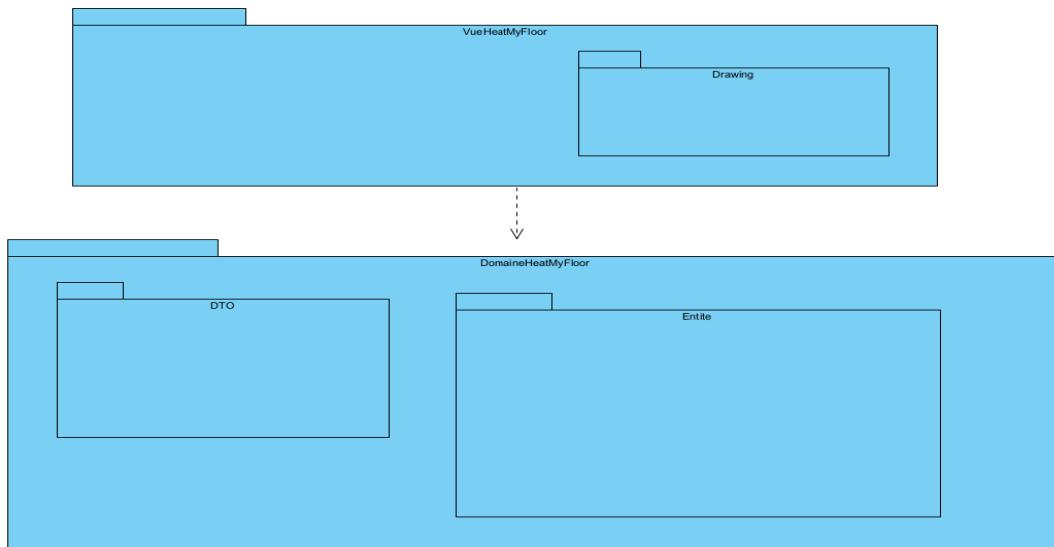
Le diagramme est bien structuré, avec une séparation claire des responsabilités entre les couches DTO, Domaine et Vue/Contrôleur. Le formalisme UML est respecté, et le niveau de détail est suffisant pour comprendre l'architecture et les interactions principales du système.

Conclusion

Le design proposé est cohérent et respecte les contraintes énoncées. L'application du principe du contrôleur de Larman via la classe **HeatMyFloorController** assure une bonne séparation des préoccupations et une gestion centralisée des requêtes système. La classe **DrawingPanel**, assistée par **MainDrawer**, gère efficacement l'affichage graphique. L'utilisation des DTO facilite la communication entre les couches, et les relations d'agrégation/composition sont correctement représentées, contribuant à un design robuste et maintenable.

3 Architecture logique

3.1 Diagramme de packages



Ce package contient les classes responsables de l'affichage et de l'interface graphique :

1. Vue

- **DrawingPanel**
 - Contient une référence à *MainWindow*.
- **MainWindow**
 - Contient une référence vers *DrawingPanel*.
 - A un contrôleur : *HeatMyFloorController*
- **MainDrawer (du package Drawing)**
 - Classe utilitaire pour dessiner les pièces et les objets sur l'interface graphique.
 - Attributs : *drawPiece()*, *drawThermo()*, etc.

2. DTO (Data Transfer Objects)

Ce package sert à transporter des données entre les couches (ex : Vue ↔ Domaine) sans logique métier :

- **MembraneDTO**
 - Contient des distances, intersections, et une *Polygon*.
- **IntersectionDTO**

- Représente une intersection avec une position et des précédents.
- **PieceDTO**
 - Représente une pièce avec forme, meubles, éléments chauffants, etc.
- **MeubleDTO**
 - Contient les données d'un meuble (forme, nom, drain, etc.).

3. Domaine

Le cœur du métier de l'application. Il contient la logique métier et les entités principales :

- **Entités principales (package Entité) :**
 - **Pièce** : représente une pièce contenant meubles, membrane, thermostat, etc.
 - **Meuble** : classe abstraite ou parent d'objets comme *MeubleAvecDrain*, *MeubleSansDrain*.
 - **Thermostat**
 - **Membrane**
 - **ElementChauffant**
 - **Intersection**
- **Classes associées :**
 - **Constraints** : constantes utilisées pour les vérifications.
 - **HeatMyFloorController** : le contrôleur principal de l'application. Contient la logique de création, modification, suppression des objets (meubles, membranes, éléments chauffants...).

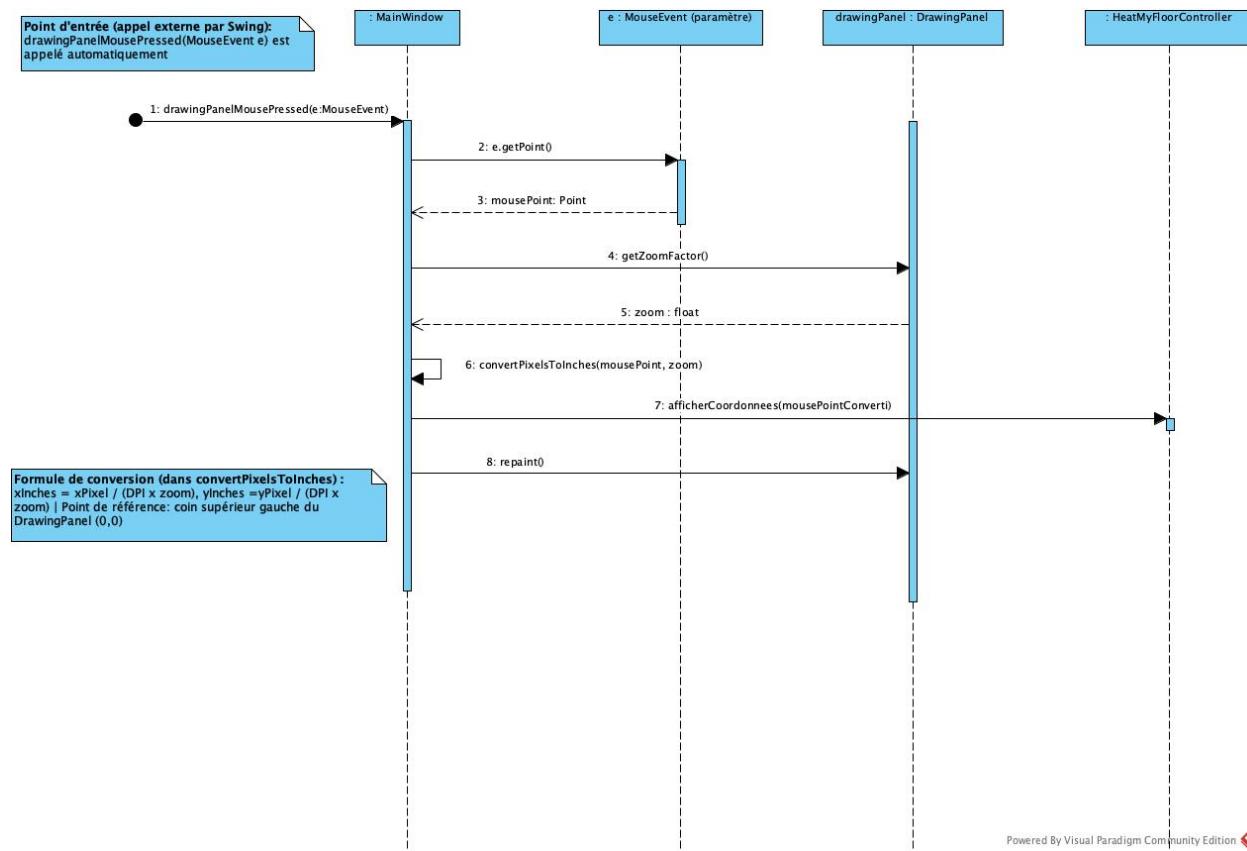
4 Relations entre packages

- **Vue ↔ Drawing :**
 - *MainDrawer* est appelé et utilisé par la classe *DrawingPanel* afin de dessiner les différents objets de notre application
- **Vue ↔ Domaine :**
 - *MainWindow* et *HeatMyFloorController* utilisent des objets du domaine pour les manipulations.
 - *DrawingPanel* interagit avec *PieceDrawer*, qui utilise les entités du domaine pour dessiner.
- **Domaine ↔ DTO :**
 - Le contrôleur (*HeatMyFloorController*) convertit des objets DTO en objets métier (ex : *MeubleDTO* → *Meuble*).
 - Les entités comme *Pièce*, *Membrane*, *Intersection*, utilisent des versions DTO pour l'échange de données.

4 Diagrammes de séquence de conception (DSC)

4.1 Détermination de l'élément sur lequel a lieu un clic de souris

4.1.1 Déterminer les coordonnées (x, y) en pouces lors d'un clic de souris



Description: Ce diagramme illustre le traitement effectué automatiquement par l'interface Swing lors d'un clic de souris dans le panneau de dessin (*DrawingPanel*). L'évènement *MouseEvent* est capturé par la méthode `drawingPanelMouseClicked(MouseEvent e)` de la classe *MainWindow*.

1. Réception de l'évènement:

L'appel `drawingPanelMouseClicked(e)` est déclenché par le système d'évènements de Java (Swing) après un clic de la souris sur la zone du *DrawingPanel*. Par contre, l'objet *DrawingPanel* a été préalablement abonné aux évènements de clic de souris et ajusté pour exécuter cette fonction. Ainsi, cet appel représente le point d'entrée externe du diagramme.

2. Récupération du point de clic :

L'objet *MouseEvent* fournit les coordonnées du clic en pixels via la méthode `e.getPoint()`. Ce point est stocké dans la variable `mousePoint`.

3. Récupération du facteur de zoom :

La fenêtre principale (*MainWindow*) appelle ensuite la méthode *getZoomFactor()* du *DrawingPanel* afin d'obtenir la valeur du zoom actuel appliquée à l'affichage.

4. Conversion des coordonnées :

Les coordonnées en pixels sont converties en pouces à l'aide de la méthode *convertPixelsToInches(mousePoint, zoom)*. Cette conversion tient compte à la fois de la résolution de l'écran et du niveau de zoom pour obtenir une position réelle sur le plan.

5. Affichage des coordonnées converties :

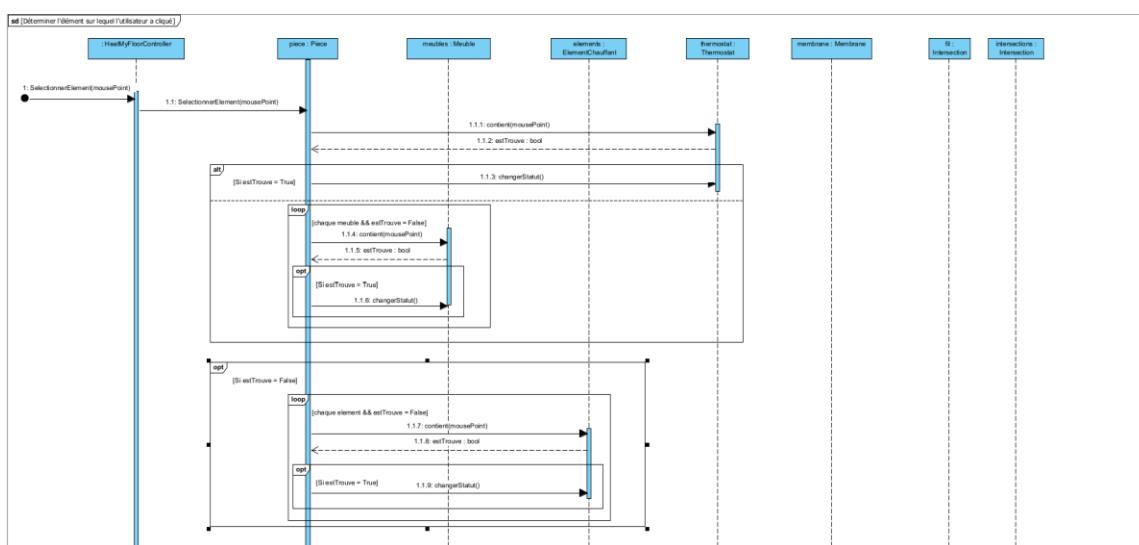
Les coordonnées sont converties et passées à un objet point : *mousePointConverti* et celles-ci sont ensuite affichées sur l'interface à l'aide de la méthode *afficherCoordonnees()* du contrôleur.

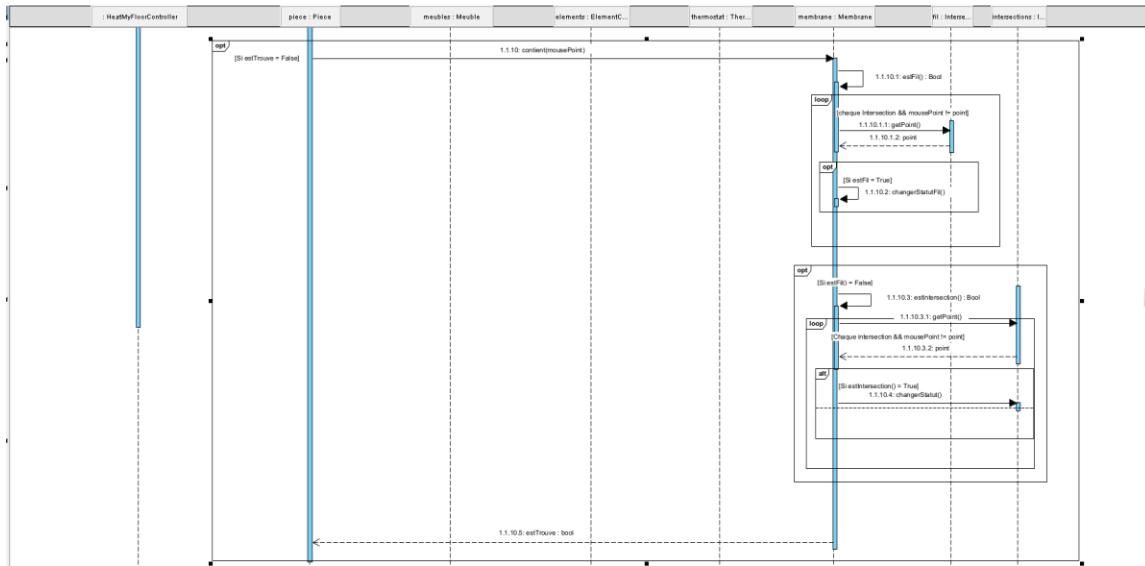
6. Rafraîchissement du panneau :

Enfin, un appel à *repaint()* est effectué pour mettre à jour l'affichage du *DrawingPanel* après la prise en compte du clic.

Finalement, ce diagramme montre comment l'application transforme un clic de souris exprimé en pixels en coordonnées physiques (en pouces) et de l'affichage de celles-ci. Cette étape est essentielle pour l'obtention des coordonnées des différents objets de notre application en tenant compte du zoom ou de la résolution d'affichage.

4.1.2 Déterminer l'élément sur lequel a lieu le clic





Description: Ce diagramme commence par un appel au contrôleur principal (*HeatMyFloorController*) avec les coordonnées du clic déjà converties en pouces. L'objectif est d'identifier sur quel élément (meuble, élément chauffant, thermostat, membrane, fil, intersection, etc.) l'utilisateur a réellement cliqué.

1. Appel du contrôleur :

La méthode *SelectionnerElement(mousePoint)* est invoquée dans le contrôleur. Elle reçoit en paramètre les coordonnées du point du clic de la souris.

2. Recherche dans la pièce :

Le contrôleur appelle la méthode *SelectionnerElement(mousePoint)* sur l'objet *Piece*, qui gère la liste des différents objets graphiques présents et lui passe le point en argument (meubles, éléments chauffants, thermostats, etc.).

3. Parcours des meubles:

Une boucle parcourt tous les objets de type *Meuble*:

- La méthode *contient(mousePoint)* vérifie si le point de clic se trouve à l'intérieur du contour du meuble.
- Si un meuble est trouvé (*estTrous* = True), sa méthode *changerStatut()* est appelée pour changer le statut de l'objet comme objet sélectionné.

4. Parcours des éléments chauffants:

Si aucun meuble n'est sélectionné, la recherche se poursuit sur les objets de type *ElementChauffant* selon la même logique (boucle + test *contient(mousePoint)* + appel *changerStatut()* si trouvé).

5. Recherche sur le thermostat, les éléments formant la membrane (fil et intersections) :

- Le diagramme montre des blocs alternatifs (alt) et optionnels (opt) qui parcourent chaque catégorie d'objets.

- Pour les intersections, le diagramme descend jusqu'au niveau des points associés (getPoint()) afin de vérifier précisément si le clic correspond à une intersection donnée.

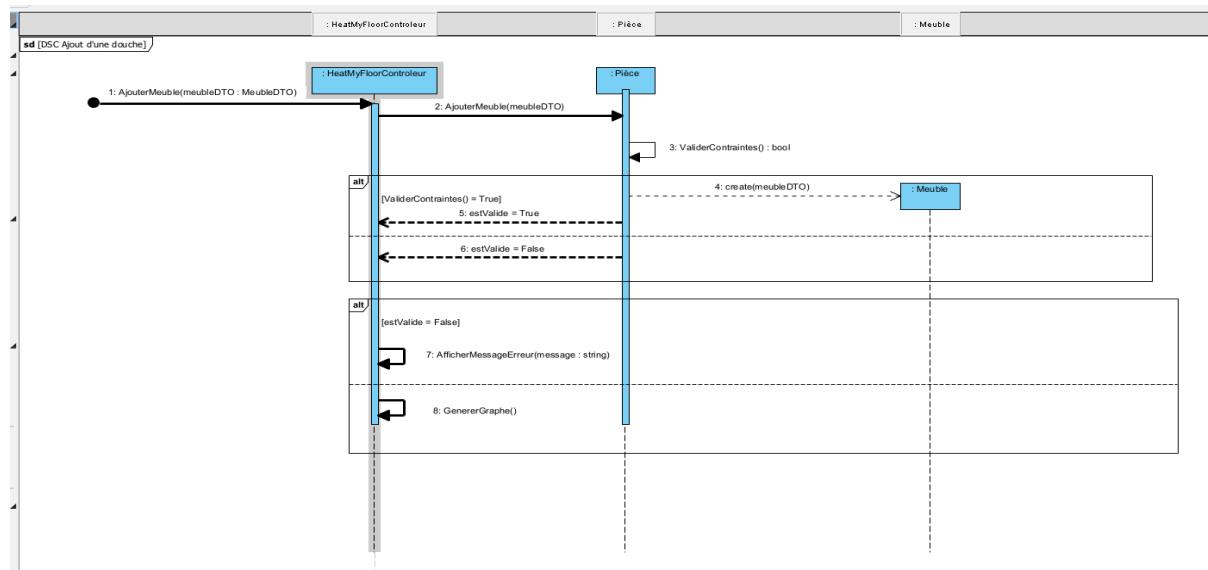
6. Seuil de détection:

Une note indique qu'un seuil de tolérance (0.1 à 0.2 pouces) est appliquée pour permettre une détection plus réaliste, évitant d'exiger un clic parfaitement exact sur un pixel.

Pour finir, ce diagramme décrit la logique de recherche hiérarchique permettant d'identifier l'objet exact sur lequel l'utilisateur a cliqué en changeant son statut. Ainsi le travail de matérialisation de l'objet sélectionné est laissé au DrawingPanel afin d'appliquer un style différent à l'objet selon son statut.

Le contrôleur délègue le travail de recherche à la pièce qui interroge successivement chaque type d'élément garantissant ainsi un comportement précis et extensible.

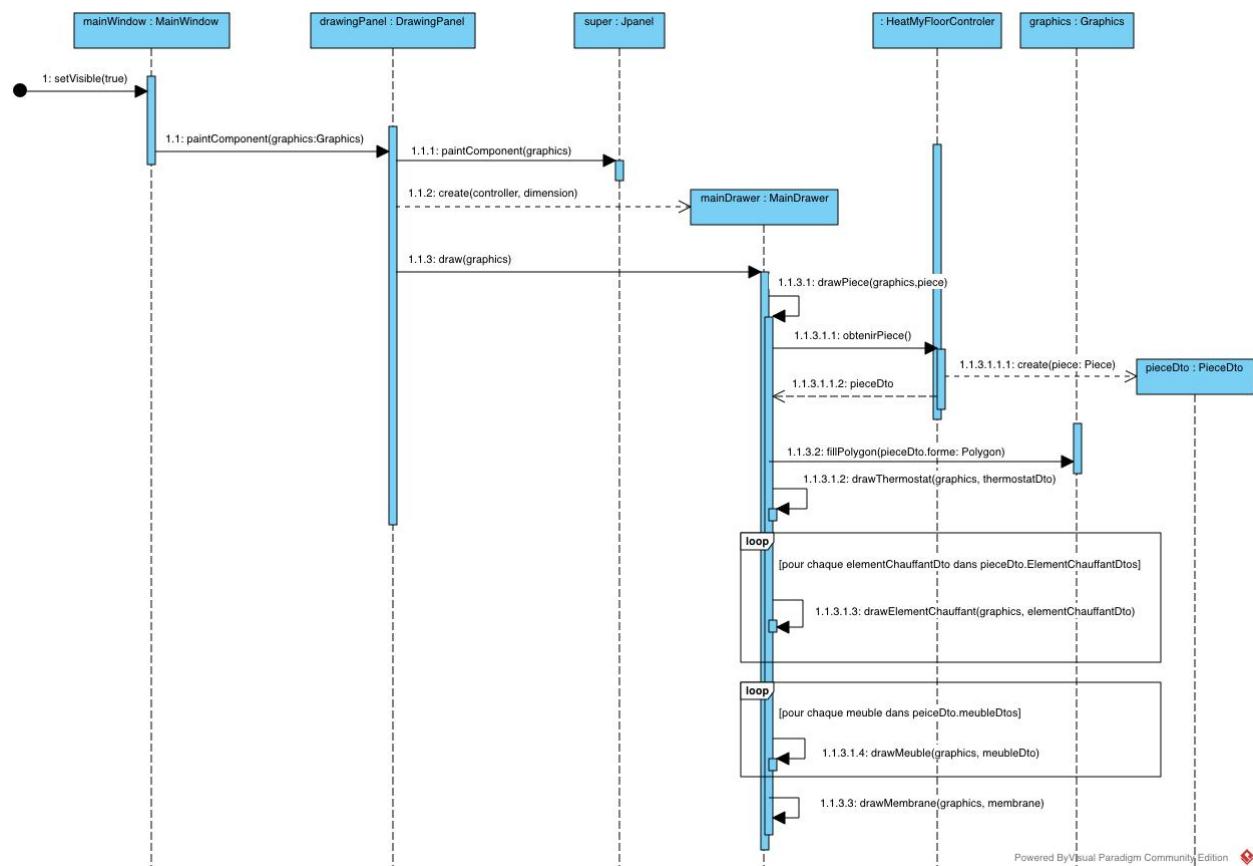
4.2 DSC Ajout d'une douche



Description: Diagramme qui illustre les actions effectuées par le système lors de l'ajout d'une douche.

1. Les informations nécessaires à la création d'un meuble sont passé en paramètres au contrôleur.
2. Le contrôleur envoie l'information sous forme de *MeubleDTO* à la classe pièce à l'aide de la méthode *AJouterMeuble*.
3. La pièce valide les contraintes pour s'assurer que le meuble peut être ajouté.
4. Si le meubleDTO remplit les conditions et valide les contraintes, alors un meuble est créé avec son constructeur et en argument le meubleDTO et ajouté à la liste des meubles existants.
5. Un statut "True" est retourné au contrôleur si tout est ok
6. Un statut "False" est retourné au contrôleur si non conforme
7. La méthode d'affichage de message d'information ou d'erreur dans la zone dédiée est appelée, car l'ajout a été invalide
8. Après avoir confirmé la création de la douche, la méthode *GénérerGraphe()* est déclenché pour recalculer le chemin du fil.

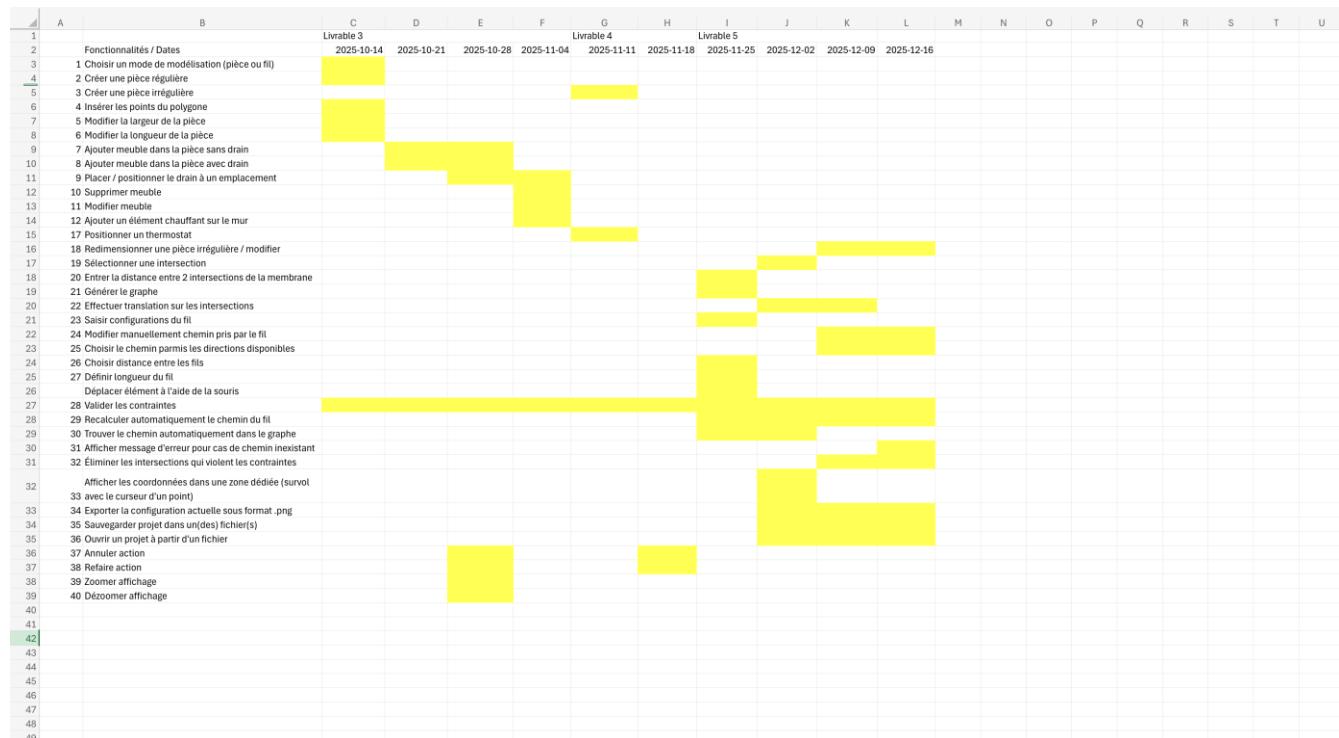
4.3 DSC Affichage de la vue



Description: Diagramme qui illustre l'affichage de la vue de notre application

1. Le mainWindow va appeler sa méthode setVisible() et l'initialiser avec le Boolean true pour afficher la base de notre application (1).
 2. Le drawingPanel va appeler une méthode paintComponent() avec un paramètre graphics (1.1).
 3. Notre drawingPanel va abord appeler la méthode super du JPanel (1.1.1).
 4. Le mainDrawer va créer notre conteneur ou les éléments seront visible (1.1.2).
 5. Le conteneur étant en place il va commencer à dessiner les éléments (1.1.3).
 6. Le mainDrawer va s'appeler avec une méthode pour dessiner une pièce (1.1.3.1).
 7. On va demander au contrôleur la pièce il va à son tour créer un objet Dto (pieceDto).
 8. Le contrôleur retourne au mainDrawer une pieceDTO.
 9. On va afficher une pièce avec une forme régulier par défaut (1.1.3.1.2).
 10. On va dessiner son thermostat (1.1.3.1.2).
 11. Pour chaque élément chauffant présent dans la pièce on dessine l'élément chauffant (1.1.3.1.3).
 12. Pour chaque meuble présent dans la pièce on dessine le meuble (1.1.3.1.4).
 13. On dessine aussi la membrane dans la pièce.

5 Plan de travail mis à jour (Gantt)



6 Contribution des membres de l'équipe

Membres de l'équipe:	Taches réalisées:
Hadadak Ndwedeme Dilane	-Diagramme de classes de conception -Diagramme de gantt -Diagramme de packages -1ere Version compilable
Sayi Camara	-Diagramme de classes de conception -Diagramme de gantt -Diagramme de packages -1ere Version compilable
Staëlle Eugène	-Diagramme de classes de conception -Diagramme de gantt -DSC 3.1.1 et 3.1.2 (Élément clic souris) -Texte explicatif du diagramme de classes de conception
Charles-Auguste Plouffe	-Diagramme de classes de conception -Diagramme de gantt -DSC 3.2 (ajout de douche) -Tableau de contribution
Jacky Masson Kemajou de Tonga	-Diagramme de classes de conception -Diagramme de gantt -DSC 3.3 (Affichage de la vue)