IUT Belfort-Montbéliard

UNIVERSITÉ DE FRANCHE-COMTÉ

INFO DUT Informatique IUT Belfort-Montbéliard

THALES

Réalisation d'un outil de prototypage IHM pour composants

RAPPORT DE STAGE

Du 06 avril au 11 juin 2021

Lisa DOYEN

Entreprise : Thales DMS Brest

DUT Informatique
2ème année

Tuteur en entreprise :
Pierre LABORDE
Senior UX & UI designer

Tuteur enseignant :
Jean-Claude CHARR
Enseignant

Remerciements

Durant les 10 semaines de travail sur ce projet, différents intervenants ont contribué à la compréhension du contexte du système, au développement et au résultat final présenté dans ce rapport, il est important de les en remercier :

En premier lieu, Pierre Laborde, senior UX & UI designer au sein de l'équipe UX & prototypage et encadrant de ce projet pour son aide, sa patience et sa disponibilité. Son écoute et ses conseils m'ont permis de répondre au mieux à la problématique donnée, et de trouver des solutions en adéquation avec les objectifs du projet.

Ensuite, les autres membres de l'équipe UX & prototypage pour leur accueil, leur gentillesse et leur aide.

Pour finir, je tiens à remercier Jean-Claude Charr, professeur à l'IUT de Belfort-Montbéliard pour son implication dans le projet en tant que tuteur enseignant.

Toutes ces personnes se sont rendues disponibles pour me permettre de réaliser ce projet dans d'excellentes conditions.

Sommaire

Introduction

Ce présent rapport est le fruit d'un travail effectué dans le cadre d'un stage de fin de 2ème année de DUT réalisé dans l'entreprise de matériel électronique Thales DMS à Brest, au sein de l'équipe UX et prototypage et de l'Institut Universitaire de Technologie de Belfort-Montbéliard du 6 avril au 11 juin 2021 (10 semaines). Ce stage a pour objectif d'allier l'application des connaissances théoriques et pratiques vues au cours de cette formation à une première expérience professionnelle.

L'objectif de ce stage est de réaliser un outil de prototypage IHM avec les solutions de développements Pharo, en open source et sous la licence *Massachusetts Institute of Technology* (MIT). Il met en œuvre un processus de conception s'inspirant du *Design Thinking* et des méthodes UX Design. Les rédactions de documents, de tests unitaires et de rapports de bugs sont également des tâches à effectuer durant ce stage.

Les utilisateurs de cet outil sont les membres de l'équipe UX & prototypage, leur besoin est d'avoir à disposition un outil sur des composants. Il permettrait de modéliser un ensemble de composants. Ces derniers sont utilisés dans le cadre de leurs activités.

Aucune application n'a été réalisé auparavant, cependant, quelques ébauches et idées étaient déjà d'actualité. Ce stage représente un intérêt très important en raison de la totale nouveauté du langage pour moi, de l'environnement de travail et des méthodes d'évolution du projet. Ainsi, il m'apporterait de nouvelles connaissances et de nouvelles compétences ce qui fait l'objet d'un avantage pour la suite de ma carrière. Le développement d'un outil graphique est également un élément qui a motivé mon envie et mon souhait d'effectuer ce stage au sein de Thales.

Ce rapport est construit sur 5 parties afin de présenter l'entreprise, les différentes étapes de conception et de réalisation du projet accompagnée des difficultés rencontrées, des solutions apportées et des résultats obtenus.

La première partie présente l'entreprise, l'équipe dans laquelle j'ai été intégrée ainsi que leurs missions principales. La deuxième partie présente le sujet du stage avec son contexte précis avec ses avantages et ses inconvénients. La troisième partie décrit la

démarche du projet et retrace le déroulement de mes missions effectuées qui constitue un cahier des charges. La quatrième partie porte un regard critique sur le travail réalisé. La cinquième et dernière partie annonce un bilan du stage contenant le bilan pour l'entreprise, le bilan humain et le bilan pédagogique. Enfin, une conclusion clôturera ce rapport.

1 Présentation de l'entreprise

1.1 Le groupe THALES

THALES est un groupe international leader mondial dans les systèmes d'informations critiques.

Les cinq principaux domaines de compétence de cette entreprise sont : l'Identité et Sécurité numérique, la Défense (Air, Terre, Naval et Interarmées) et la Sécurité, l'Aéronautique, l'Espace et le Transport terrestre. Ces différents marchés représentent 17,0 milliards d'euros de chiffre d'affaires en 2020. THALES est implanté dans 68 pays et compte 82000 collaborateurs.

Figure 1 : Secteurs d'activités de Thales

Dans le domaine de l'Aéronautique et de l'Espace, THALES intervient sur tous les grands programmes aéronautiques civils et militaires et offre des solutions spatiales de bout en bout.

Dans les Systèmes aériens, l'entreprise propose des solutions globales de sécurité et de surveillance de l'espace aérien civil et militaire. Thales s'impose comme l'un des acteurs majeurs dans les Systèmes Terre et Interarmées mais aussi dans le Naval par sa coopération renforcée avec *Naval Group*. Le groupe répond aux besoins de sécurité de nombreuses entreprises et offre des services extrêmement diversifiés. De plus, THALES s'affirme comme

un acteur important dans les domaines de recherche & développement pour les équipements et les systèmes avec l'intégration de ceux-ci, et contribue autant aux marchés du domaine civil que militaire.

La mission de THALES est d'aider ses clients à prendre les bonnes décisions à chaque moment décisif en combinant deux piliers. Le premier est l'expertise technologique et des décennies d'expérience au service des cinq domaines de compétences très exigeants. Le second est l'offre de technologies, axée sur la "chaîne de décision critique" suivante :

- La détection et la collecte de données ;
- La transmission et le stockage des données ;
- Le traitement des données et la prise de décision.

THALES a investi massivement dans quatre technologies clés pour accélérer la transformation digitale de ses clients : l'Intelligence artificielle (IA), la Connectivité/Internet des objets (IOT), le *Big data* et la Cybersécurité. Ces technologies numériques sont appliquées dans des domaines critiques où il n'y a pas de place pour l'erreur, elles doivent être fiables à 100 % et dans des secteurs fortement réglementés.

En France, l'entreprise Thales emploie aujourd'hui 34300 personnes réparties sur 70 grands sites. Le marché de la défense est divisé en quatre secteurs d'activités : l'Air, la Terre, le Naval et l'Interarmées.

- Les systèmes Aériens :

    - Systèmes d'armes, de surveillance et de détection ;
    - Solutions radar, commandement et contrôle, radar de champ de bataille ;
    - Solutions de gestion du trafic aérien.
    - Référence :
        - Fournisseur de systèmes avioniques clés à Airbus et Eurocopter ;
        - Système de commandement et de contrôle aérien SCCOA ;
        - Missile de très courte portée *Starstreak*.

- Les systèmes Terre et Interarmées :

- Gamme complète de solutions terrestres : des grands systèmes coopératifs, systèmes soldat/véhicule aux équipements clé et services ;
- Systèmes de renseignement, commandement et communication bout-en-bout (systèmes C4ISR) interopérables, pour les opérations interarmées et info-centrées ;
- Equipements de communication et d'optronique pour les milieux air, terre et mer ;
- Référence :
    - Bulle opérationnelle aéroterrestre (BOA) ;
    - Maîtrise d'œuvre du programme Syracuse III ;
    - Système d'information et de commandement pour les forces terrestres (SICF).

- Le Naval :

- Maîtrise d'œuvre et intégration de systèmes ;
- Équipements et systèmes pour bâtiments de surface ;
- Équipements et systèmes sous-marins ;
- Services (soutien de flottes et maintien en condition opérationnelle) ;
- Référence :
    - Frégate Multi-Missions Franco - Italienne (FREMM) ;
    - Porte-avions (PA2) ;
    - Sous-marins Scorpène (Inde, Chili, Malaisie).

1.2 L'équipe UX & Prototypage

L'équipe UX & Prototypage regroupe cinq membres. Ce sont des ingénieurs logiciels avant tout.

Dans le cadre de mon stage, j'ai pu intégrer cette équipe et découvrir son fonctionnement, ses activités et sa structure.

L’équipe réalise l’expérience utilisateur (UX) et l’interface utilisateur (UI).

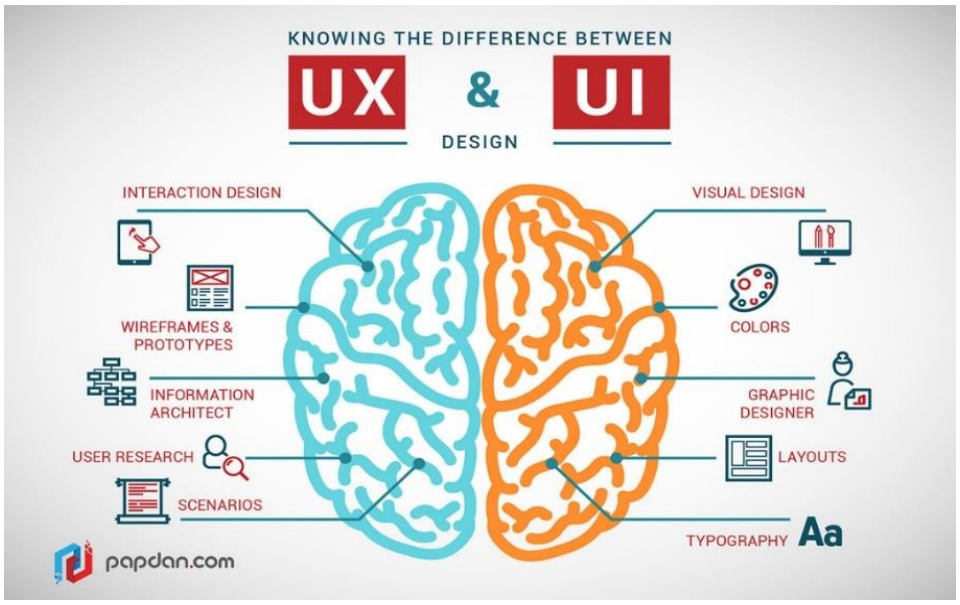

Figure 2 : Schéma définissant l'UX et l'UI design
Source : voir bibliographie

Le terme UI (acronyme de l’anglais : *User Interface*), signifie interface utilisateur en français. L’interface utilisateur est la présentation graphique d’une application. Il comprend les boutons, le texte, les images, les curseurs, les champs de saisie de texte et tous les autres éléments avec lesquels l’utilisateur interagit.

Le terme UX (acronyme de l’anglais : *User eXperience*), expérience utilisateur en français, désigne la qualité de l’expérience vécue par l’utilisateur dans toute situation d’interaction. Il s’agit d’évaluer le ressenti et les émotions de l’utilisateur lors de l’utilisation d’une interface, d’un appareil digital ou plus largement en interaction avec tout dispositif ou service.

Un UX designer décide comment l’interface utilisateur fonctionne, tandis que l’UI designer décide de l’apparence de l’interface utilisateur.

Ainsi, il faut :

- Comprendre le besoin utilisateur (son profil) ;
- Définir l’ergonomie et la structure ;
- Elaborer le graphisme et le design ;

- Et expérimenter ces 3 points précédents.

Les compétences principales à avoir en tant qu'UX/UI designer sont :

- Psychologie ;
- Ergonomie ;
- Artistique ;
- Ethnologie.

De plus, une des méthodes utilisées est le *design thinking* permettant d'appréhender l'expérience utilisateur. Cette méthode est découpée en plusieurs étapes :

- L'empathie ;
- La définition ;
- L'idéation ;
- Le prototype ;
- Et le test

1.3 Les activités de l'équipe

L'équipe UX et prototypage remplit différentes missions.

Tout d'abord, ils sont en interface étroite avec les clients et les utilisateurs afin de capter et comprendre le besoin. La 1ère étape comprend l'organisation de groupes de travail (GT) afin d'effectuer des brainstormings et des ateliers pour pouvoir amener et échanger différentes idées.

La 2ème étape consiste à élaborer des maquettes plus précisément désignées par les termes anglais le « *WireFraming* » ou le « *Sketching* ». Cette phase peut contenir un certain nombre d'échanges entre l'équipe et les utilisateurs afin de converger naturellement vers la solution idéale. Les logiciels graphiques exploités sont PicPick, *Pencil* et Yed.

Ensuite, vient l'étape du prototypage[1] d'Interface Homme-Machine (IHM). Chez Thales, les activités de prototypage d'interface homme-machine représentent une partie très importante du processus de production logiciel. Le prototypage industriel IHM est la construction de prototypes logiciels aussi proche que possible du produit réel du point de vue IHM. À l'aide de ceux-ci, l'équipe évalue la conception du logiciel IHM et expérimente des cas d'utilisation complets avec les utilisateurs finaux. Grâce au prototypage, les besoins et les problèmes architecturaux sont rencontrés et résolus avant le début du développement logiciel industriel du produit final.

Certains des prototypes de Thales mettent en œuvre un comportement commercial complet et remplissent des exigences fonctionnelles. Ainsi, ce sont des prototypes qui évolue au fil du temps. Les ingénieurs récupèrent certaines parties et les réutilise dans d'autres projets.

Enfin, Le prototypage est suivi par une phase d'industrialisation au sein d'une autre équipe. Durant cette phase, les produits finaux sont construits en se basant sur les prototypes. Les ingénieurs systèmes au sein de l'équipe d'ingénierie font le lien entre les entités matérielles, les entités logicielles et les utilisateurs.

Afin de capitaliser et favoriser la réutilisation d'éléments de projets, l'équipe met en œuvre des éléments unitaires d'IHM. Ces éléments peuvent prendre des formes plus ou moins complexes. Ils sont regroupés en sous-systèmes de composants. Les prototypes sont constitués de composants qui peuvent soit être issus d'un existant (en l'état ou adapté) ou tout simplement nouveaux.

Les architectures à base de composants apportent la modularité nécessaire pour permettre la réutilisation et l'évolution des prototypes. De plus, selon les équipes, les propriétés descriptives des modèles de composants permettent de communiquer facilement entre les différentes équipes et donc la capitalisation via le partage de connaissances se met en place.

[1] **Prototypage** : Le prototypage est la démarche qui consiste à réaliser un prototype. Ce prototype est un exemplaire incomplet et non définitif de ce que pourra être le produit ou l'objet final. Il existe deux dégrées de prototypage IHM selon le niveau d'interactivité : le prototypage horizontal (partie graphique de l'interface) et le prototypage verticale (met en œuvre les fonctionnalités afin que l'utilisateur puisse dérouler un scénario complet d'utilisation). (Source : *Wikipédia*)

Pour construire ces prototypes, l'équipe utilise de nombreux langages de programmation comme VisualWorks, SmallTalk, Pharo, Java et C++.

Thales utilise SmallTalk depuis 2005. Ce langage est rapide à mettre en œuvre et efficace concernant le design et la programmation de prototypes complexes. Sa capacité à modifier de façon vivante un design directement devant les clients est également un point fort.

L'image ci-dessous liste les outils utilisés par l'équipe.

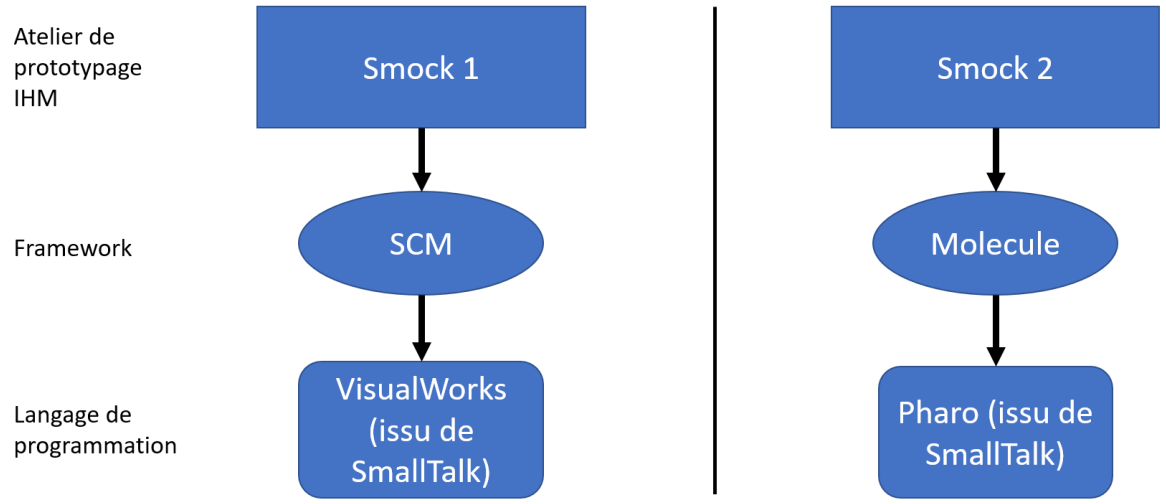

Figure 3 : Schéma des outils utilisés par l'équipe

Les *Frameworks*[2] utilisés sont Molecule avec le langage Pharo et SmallTalk Component Model (SCM) avec le langage VisualWorks. Les ateliers de prototypages IHM sont Smock 1 (pour VisualWorks) et Smock 2 (pour Pharo).

[2] ***Frameworks*** : en programmation informatique, un *Framework* (appelé aussi infrastructure) désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). (Source : *Wikipédia*)

1.4 Activités de développement de l'équipe : Les Composants

1.4.1 Description

1.4.1.1 Molecule

L'équipe utilise Molecule depuis 2016. Il s'agit d'un *Framework open-source*[3] orienté composant[4] pour Pharo, sous licence de l'Institut de Technologie du Massachusetts (MIT)[5]. Pharo est un langage de programmation dynamique[6] orienté objet[7] et sous licence MIT créé en 2009. Il est largement inspiré de SmallTalk.

Figure 4 : Logo Molecule
Source : voir bibliographie

Figure 5 : Logo Pharo
Source : voir bibliographie

Molecule fournit la capacité de faire une architecture adaptée au développement d'interfaces utilisateur graphique, basée sur des fonctionnalités, réalisées sous forme de composants.

Thales utilise Molecule pour capitaliser son expérience dans les développements des composants. Aussi, les caractéristiques de Pharo permettent de modifier dynamiquement les architectures de composants au moment de l'exécution lors de démonstrations avec les utilisateurs finaux.

---

[3] ***Open-source :*** code source ouvert en français, s'applique aux logiciels dont la licence respecte des critères précisément établis par *l'Open Source Initiative*, c'est-à-dire les possibilités de libre redistribution, d'accès au code source et de création de travaux dérivés. (Source : *Wikipédia*)

[4] **Orienté composant :** La programmation orientée composant (POC) consiste à utiliser une approche modulaire de l'architecture d'un projet informatique, ce qui permet d'assurer au logiciel une meilleure lisibilité et une meilleure maintenance. (Source : *Wikipédia*)

[5] **Licence MIT** : La licence MIT est une licence de logiciel pour logiciels libres et open source, provenant de l'Institut de Technologie du Massachusetts à la fin des années 1980. (Source : *Wikipédia*)

[6] **Langage de programmation dynamique** : en informatique, le terme langage de programmation dynamique décrit des langages de haut niveau qui exécutent au moment de l'exécution des actions que d'autres langages ne peuvent exécuter que durant la compilation. (Source : *Wikipédia*)

[7] **Orienté objet** : La programmation orientée objet (POO), ou programmation par objet, est un paradigme de programmation informatique. Elle consiste en la définition et l'interaction de briques logicielles appelées objets. (Source : *Wikipédia*)

Le modèle de composant utilisé par Molecule s'inspire du *Lightweight Corba Component Model* (*Lightweight* CCM)[8]. Il permet la spécification des composants selon le standard *Common Object Request Broker Architecture* (CORBA)[9].

1.4.1.2 Composant

Un composant est une classe[10] instanciée [11]qui contient des interfaces. Une interface est un ensemble de méthodes (fonctions) accessibles depuis l'extérieur de cette classe.

1.4.1.2.1 Interfaces

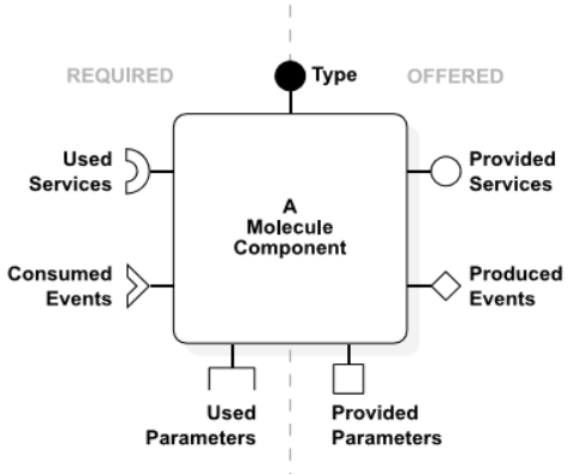

Figure 6 : Vue du Composant Molecule

Un composant va pouvoir offrir des interfaces et/ou recevoir des interfaces. Parmi celles qui sont offertes (la partie « *Offered* » de la figure 5), il peut fournir un service (*Provided Services*), produire un évènement (*Produced Events*) et produire un paramètre (*Provided Parameters*). Parmi celles qui sont reçues (la partie « *Required* » de la figure 5), il peut utiliser un service (*Used Services*), consommer un évènement (*Consumed Events*) et utiliser un paramètre (*Used Parameters*). Pour faire le parallèle avec un exemple concret, une cafetière

[8] ***Lightweight Corba Component Model* (*Lightweight* CCM)** : un ensemble de norme qui propose de décrire un modèle de composants simplifié en reprenant les principes des normes de composants CORBA. (Source : *Wikipédia*)

[9] ***Common Object Request Broker Architecture* (CORBA)** : norme qui définit une architecture logicielle pour le développement de composants. Ces composants sont assemblés afin de construire des applications complètes. Ils peuvent êtres écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes. (Source : *Wikipédia*)

[10] **Classe** : En programmation orientée objet, une classe déclare des méthodes et des propriétés (attributs). (Source : *Wikipédia*)

[11] **Instance** : Une instance est un exemplaire d'un même composant. Il peut y avoir plusieurs instances du même composant dans un projet.

est un composant, elle fournit du café (service). Lorsque la cafetière est vide, elle le signale (évènement).

1.4.1.2.2 Contrat et *Type*

Le composant établit un contrat défini par un *Type* contenant l'ensemble de ses interfaces. Le *Type* définit les interfaces comme les services fournis et les événements produits par le composant. Ensuite, d'autres composants utilisent ses services fournis et/ou consomment ses évènements produits. En reprenant l'exemple précédent, une tasse est un deuxième composant. Elle va utiliser le service fourni par la cafetière (le café). La plaque chauffante est un troisième composant. Cette plaque va écouter l'événement de la cafetière (cafetière vide) afin de se couper.

1.4.1.2.3 *Trait*

L'une des particularités de Pharo est la présence des *Traits*. Un *Trait* est un ensemble de méthodes qui décrit un comportement. Chaque classe utilisant un *Trait* va automatiquement bénéficier de ses méthodes et de ses variables (classes ou instances). Un *Trait* peut également utiliser d'autres *Traits*.

Leur rôle est donc de fournir un comportement à toutes les classes utilisant ce *Trait*, indépendamment de la hiérarchie des classes. Pour l'équipe, ils sont utiles pour la réutilisation des composants.

Ces *Traits* vont être utilisés dans la création des composants et des contrats.

Il existe deux chemins pour créer un composant (voir figure 7) :

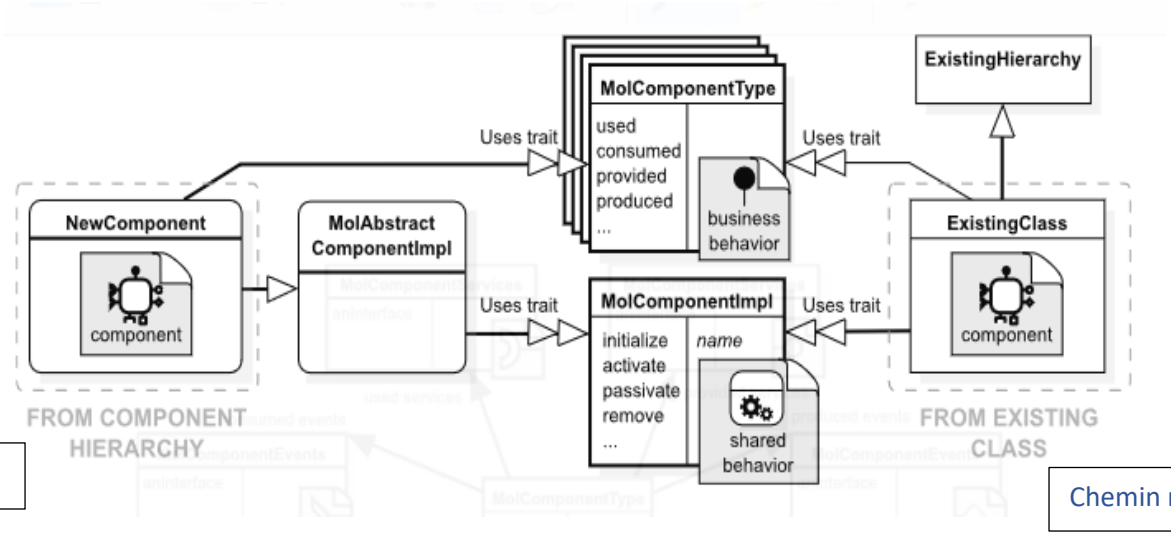

Figure 7 : Deux chemins pour créer un composant

- Dans le premier cas (en partant du chemin n°1 sur la figure 7), le composant hérite de la classe *MolAbstractComponentImpl*. *MolAbstractComponentImpl* utilise Le *Trait* appelé *MolComponentImpl* permettant d'obtenir le comportement partagé par tous les composants. Aussi, le composant utilise un *Trait* hérité du *Trait* nommé *MolComponentType* qui définit le *Type* du composant ;
- Dans le deuxième cas (en partant du chemin n°2 sur la figure 7), n'importe quelle classe existante est transformée en composant. Cette classe existante utilise le Trait *MolComponentImpl* et un *Trait* hérité du *Trait MolComponentType*. Ces deux *Traits* sont ceux mentionnés dans le premier chemin. Cette classe devient utilisable dans une application de composant Molecule tout en restant entièrement compatible avec les applications sans composant.

Le schéma visible en figure 8 récapitule la définition du contrat du composant Molecule.

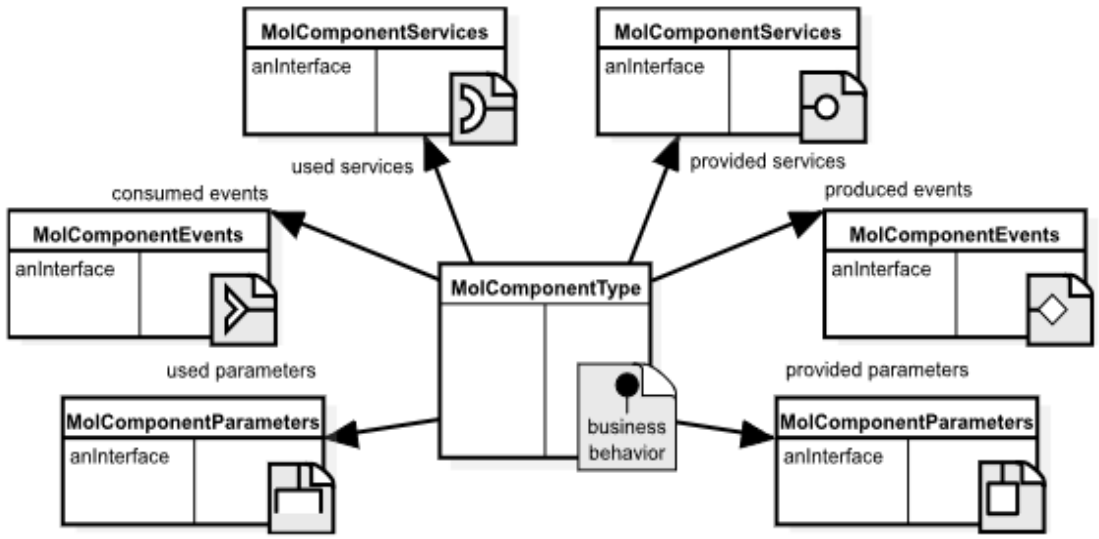

Figure 8 : Établissement du contrat du composant Molecule

Le *Type*, correspondant au carré au milieu de l'image (*MolComponentType*), d'un composant est lui-même défini comme un *Trait* (étiquette « *business behavior* »). Ce *Type* va permettre d'établir le contrat contenant les interfaces (les six autres rectangles sur la figure 8). Ces interfaces (services, paramètres et évènements) sont également définies comme des *Traits* (étiquette contenant les différentes formes géométriques).

1.4.1.2.4 Instances et cycle de vie des composants

Le composant peut être identifié par un nom spécifique. Différentes instances de ce composant peuvent donc être créées. Elles seront différenciées par leur nom.

Une instance d'un composant possède un cycle de vie comprenant quatre états possibles : *Initialized*, *Activated*, *Passivated* et *Removed* (visible en figure 9).

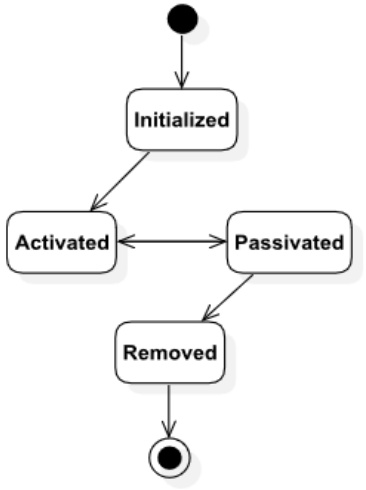

Figure 9 : Cycle de vie du composant

Après son initialisation, elle peut être activée. Par la suite, elle peut être désactivée et/ou réactivée. Enfin, elle peut être supprimée. Si un composant est désactivé, ses services ne sont plus disponibles. Ce cycle de vie sert au développeur pour maîtriser les différentes phases d'utilisation de son instance de composants

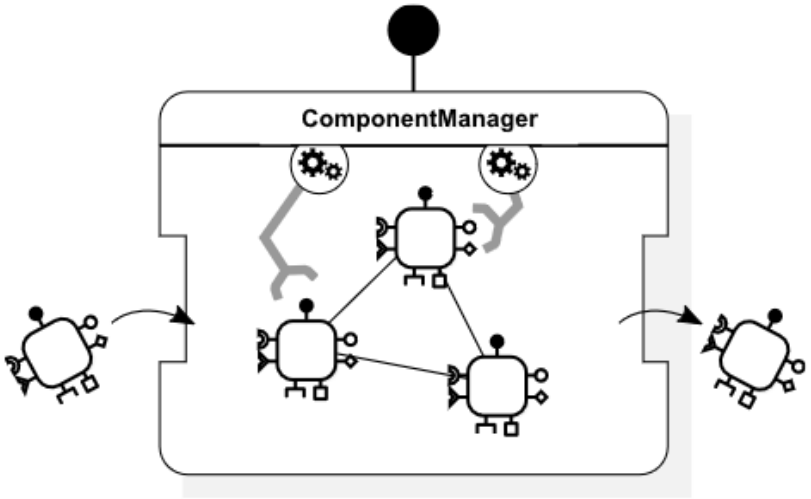

Figure 10 : ComponentManager gère l'ensemble des composants

Toutes ces instances sont gérées par le manager de composant appelé *ComponentManager* visible en figure 10. Il a pour rôle de connecter les instances de composants entre elles par l'intermédiaire de leurs interfaces.

1.4.2 Intérêts

Selon l'équipe, construire des architectures logicielles réutilisables et homogènes sont les deux principales motivations pour utiliser les composants.

Aujourd'hui, pour sélectionner quel composant réutiliser, l'équipe va explorer et expérimenter les composants dans le référentiel choisi, puis, à travers une exigence donnée, les interfaces des composants vont se connecter et interagir entre elles. Par la suite, l'équipe étudie cette interaction et détermine à travers leurs comportements, le composant qui sera réutilisé.

Il existe 3 scénarios de réutilisation des composants :

- scénario de réutilisation à long terme du code hérité ;

- la réutilisation de *Frameworks* non composants ;

- la réutilisation des interfaces métiers des composants.

Ces 3 scénarios permettent de construire des prototypes plus rapidement.

1.4.2.1 Premier scénario : scénario de réutilisation à long terme du code hérité

Le premier scénario forme la chaîne de réutilisation de prototypage visible dans la figure 11. Il est le plus commun dans l'activité de prototypage chez Thales.

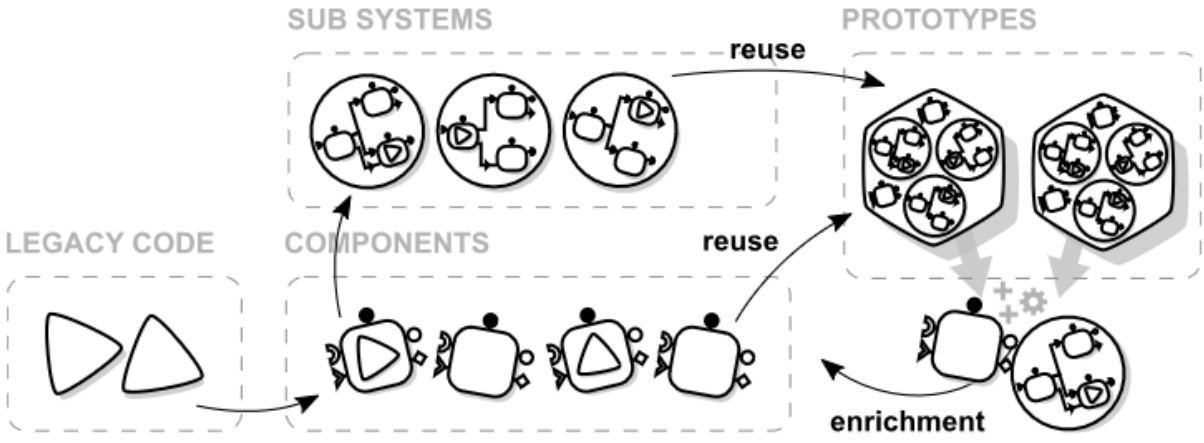

Figure 11 : Chaîne de réutilisation de prototypage

Il se traduit par la construction de composants standardisés et réutilisables à partir d'un code hérité précédent (*legacy code* sur la figure 10). L'architecture basée sur les composants aide l'équipe à la réutilisation du code hérité.

L'ensemble de ces composants forme un sous-système (*sub systems* sur la figure 10). Ce sous-système est réutilisé ultérieurement pour un autre prototype. Certains composants uniques sont également réutilisés.

Lors de chaque nouvelle construction de prototype, l'équipe identifie de nouvelles fonctionnalités ou des briques réutilisables et les réintègrent dans le référentiel des composants ou sous-systèmes existants pour une réutilisation future.

Ce type de réutilisation est désormais possible car l'équipe a capitalisé une quantité suffisante de composants.

Par exemple, cette chaîne est particulièrement vraie pour les éléments d'interfaces graphiques appelé *Panels*. Les *panels* construits à partir de composants affichent une interface publique standard. L'équipe les réutilise donc directement dans plusieurs prototypes.

Ainsi, grâce aux propriétés des composants, l'effort pour arriver aux premières versions d'un nouveau prototype sera réduit. Le nombre de nouveaux prototypes diminue aussi.

1.4.2.2 Deuxième scénario : la réutilisation de *Frameworks* non composants

Dans ce scénario, les développeurs s'appuient sur des *Frameworks open-sources* orientés objet déjà existants. Ensuite, ils transforment les classes en composants avec Molecule pour les réutiliser directement dans les prototypes. Il devient alors possible de faire interagir ces classes transformées en composant au sein des autres composants de l'application. Elles restent également utilisables dans les applications standards (chemin n°2 de la figure 7).

1.4.2.3 Troisième scénario : - la réutilisation des interfaces métier des composants

Dans le dernier scénario, le contrat et le *Type* des composants sont réutilisés en suivant la chaine de réutilisation de prototypage en figure 10.

Par exemple, prenons un prototype dans lequel il fallait migrer le système de base de données vers un nouveau système. L'accès à la base de données (c'est-à-dire les demandes de

base de données) a été créé dans un composant. L'accès aux données (c'est-à-dire les demandes de données) a été défini par le *Type* du même composant. D'autres composants communiquaient avec la base de données. L'équipe a alors pu changer le composant par un nouveau dans le prototype en conservant le contrat et le *Type*.

1.4.2.4 Difficultés

Les développeurs ne disposent pas tout le temps de toutes les connaissances pour savoir s'il faut réutiliser un composant ou non. La transmission d'informations peut se perdre et le développement peut ralentir. Aussi, rendre les composants réutilisables représente un coût.

Ainsi, il faudrait pouvoir identifier plus facilement les composants et ainsi homogénéiser les pratiques afin de faciliter le développement et par la suite, simplifier l'intégration d'éventuels nouveaux collaborateurs. Aussi, à travers cet outil de visualisation des composants, la présentation de l'activité de l'équipe à un tiers est simplifiée.

C'est à la suite de ce constat que le besoin lié au projet apparaît.

2 Présentation du sujet

Les composants sont au cœur de l'activité de prototypage de l'équipe UX & prototypage de Thales. Cependant, un outil facilitant leur visualisation manque actuellement. Le processus de création du composant est également long et répétitif et peut être amélioré. Des problèmes d'architectures liés aux composants sont également présents dans leur système.

L'objectif de ce stage est d'apporter un support graphique/visuel des composants, de pouvoir créer et modifier rapidement un composant et également de détecter et résoudre les éventuelles anomalies (anomalies techniques et architecturelles).

Les utilisateurs sont les membres de l'équipe. L'enjeu est donc de pouvoir répondre et de satisfaire leurs besoins.

Aucune application de ce type n'existe, cependant, quelques croquis avaient été réalisés.

Pour ce projet, la première phase était donc de prendre connaissance du sujet et de comprendre le besoin des utilisateurs. Des maquettes devront être réalisées puis viendra le développement de l'outil.

L'environnement de travail est en *open-source*. Mon travail est disponible sur le GitHub de Molecule.

Ce stage est un projet multi-affaires. Il contribue à la préparation et à la migration progressive de l'intégralité de l'équipe vers l'utilisation de l'atelier de prototypage avec Pharo.

3 Cahier des Charges

3.1 Démarche

Le déroulement général suit les méthodes UX et UI design.

Le projet se traduit en quatre grandes étapes :

- Une phase d'idéation qui consiste à prendre connaissance du sujet. Dans cette phase, la capture du besoin est l'élément central. Elle se fait par des interviews effectuées auprès des utilisateurs, par des workshops, des ateliers de travail avec des brainstormings. Puis, des solutions et des idées arrivent au travers de la compréhension du besoin ;

- La phase de maquettage suivra la phase d'idéation pour mettre en forme les idées évoquées. Le design apparait dans cette phase. Ainsi, des présentations de ces maquettes aux utilisateurs permettront de récolter les remarques, les éventuelles observations, ce qui permettra de faire évoluer les maquettes régulièrement vers un modèle qui emportera le consensus général ;

- Une phase de prototypage est ensuite mise en place. Dans cette phase, le travail consiste à réaliser les maquettes sous forme de prototypes ;

- Enfin, la dernière phase concerne la réalisation et le développement. Aussi, des tests utilisateurs pourront être effectués en parallèle pour pouvoir améliorer certains aspects ou pour pouvoir corriger les bugs.

3.2 Contraintes

3.2.1 Contraintes techniques

3.2.1.1 Langage de programmation : Pharo

Concernant les contraintes techniques, le langage de programmation utilisé pour ce stage est Pharo. L'équipe l'utilise depuis 2016. Ce dernier est créé le 24 avril 2008 par Stephan Ducasse et Marcus Denker, dont la syntaxe tient sur une carte postale.

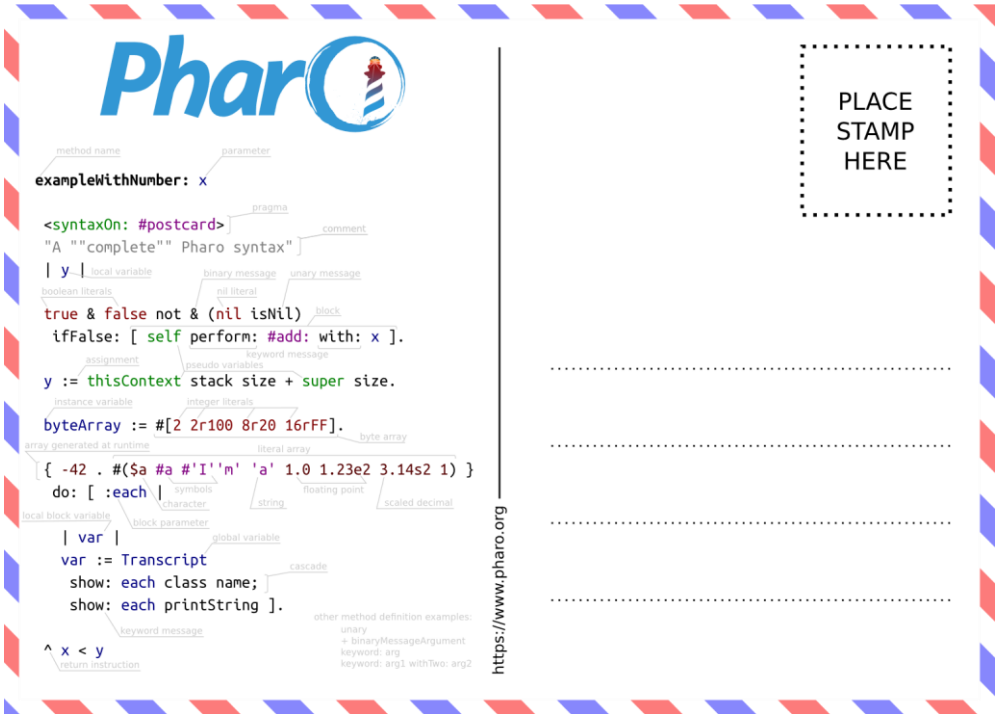

Figure 12 : Carte postale de la syntaxe Pharo
Source : voir bibliographie 2

Il est basé sur une machine virtuelle sur lesquelles sont créé des images Pharo écrites en Pharo elle-même. Concernant ses caractéristiques, tout est objet, au sens de la programmation orientée objet. Il y a de l'héritage simple c'est-à-dire qu'une classe hérite d'une autre classe. La gestion de la mémoire est automatique. Le typage est un typage dynamique c'est-à-dire que les variables peuvent prendre n'importe quelle valeur, contrairement à un typage statique ou le développeur doit indiquer de quel type est chaque variable (entier, chaine de caractères…).

Un des intérêts principaux de Pharo est qu'il n'est pas nécessaire de recompiler tout le code dans le cas de la modification d'une méthode (fonction). Il est par exemple possible de

modifier ou de créer une méthode au sein du débugueur[12] et de reprendre le flot d'exécution juste avant la modification. Certains appellent cela la méthode « *edit and continue »* au lieu de la traditionnelle méthode « *edit compile and run »*.

Concernant les ressources disponibles pour étudier ce langage, un MOOC en ligne sur *YouTube* permet de s'informer sur les principes de ce langage. De plus, une communauté très active se développe notamment via la plateforme Discord. J'ai d'ailleurs pu échanger facilement avec d'autres « *smallTalkers* ».

Cependant, l'utilisation de ce langage n'est pas très populaire. Selon Wikipédia, 600 personnes sont sur la liste de diffusion Pharo et pas plus d'une vingtaine d'entreprises utilisent aujourd'hui Pharo pour leurs développements logiciels. Ainsi, la documentation et les forums se font assez rares.

L'environnement Pharo est un langage qui ne m'a pas été enseigné à l'IUT de Belfort. Il a donc fallu prendre connaissance du langage avant de programmer. Aussi, la connaissance de Pharo des membres de l'équipe UX et prototypage est hétérogène. Certains utilisent Pharo et d'autres utilisent VisualWorks (une autre implémentation de SmallTalk).

La version utilisée pour le projet est la 9ème de Pharo. Cette version n'est pas stable (en Beta). D'après les membres de l'équipe, elle peut présenter quelques instabilités. Certains bugs sont encore présents.

3.2.1.2 Roassal3 et Spec2

Avec le langage de programmation, j'ai dû aussi apprendre Roassal3 et Spec2. Il s'agit de la 3ème version de Roassal et de la 2ème version de Spec. Ce sont des modules directement intégrés dans l'image Pharo. Roassal3 et Spec2 sont largement utilisés dans la communauté Pharo.

[12] **Debugueur** (ou débogeur) : un logiciel qui aide au développeur à analyser et corriger les bugs/erreurs du programme.

Roassal3 est un système de script pour visualisations interactives avancées, distribué sous licence MIT. Ces visualisations sont dites agiles et encouragent l'utilisation de l'environnement de développement interactif Pharo. En effet, grâce à Pharo, le développement logiciel de la création d'une visualisation est réduit par rapport aux boites à outils traditionnels. Cela peut représenter un gain de temps et d'effort. Dans le cadre du stage, cette bibliothèque graphique a été utilisée pour dessiner le composant graphiquement et les différentes relations entre eux.

Figure 13 : Logo Roassal
Source : voir bibliographie

Spec2 est un *Framework* permettant de décrire les interfaces utilisateurs. Dans le cadre de mon outil, cela se traduit par les boutons, les fenêtres ou encore les listes déroulantes. Spec2 donne la capacité de générer différentes IHM dans différentes technologies comme GTK ou Morphic. Spec2 est compatible avec Roassal3 ce qui permet d'intégrer un graphique Roassal3 dans une fenêtre Spec2.

Ce sont des modules encore en développement c'est pourquoi certaines fonctionnalités ne sont pas encore disponibles. Sur le Discord de Pharo, des channels de discussions dédiées à Roassal et Spec permettent de communiquer plus facilement.

3.2.1.3. *Open source* : utilisation de *GitHub*

L'environnement de travail était en *open source* via la plateforme collaborative *GitHub* donc l'avantage est que l'accès aux ressources d'un point de vue documentation et communication n'est pas limité. Des exemples sur Roassal3 et Spec2 sont disponibles, au sein de l'image Pharo et sur GitHub.

3.2.2 Contraintes temporelles

La phase de prise de connaissances du sujet a permis de réfléchir aux différentes fonctionnalités que l'outil pourrait intégrer. Compte-tenu de la durée du stage, il n'était pas envisageable de tout faire. Il n'y a pas eu de planning détaillé défini pour le projet car le temps de prise de connaissances et d'assimilation des outils/langages à utiliser pouvait être important.

Néanmoins, on peut découper le stage en trois parties. Environ deux/trois semaines pour la prise de connaissances et prise en main des outils, puis cinq/six semaines de développement et enfin une semaine pour rédiger les rapports et présenter le travail réalisé à l'équipe.

Ce projet a permis d'être confronté à une gestion des priorités qui est représentative d'une vraie expérience professionnelle. La phase de développement a été axée sur les fonctions prioritaires.

4 Mise en œuvre

4.1 Comprendre le besoin

4.1.1 Protocole

Après avoir pris connaissances sur le sujet, il a fallu se documenter sur le langage Pharo et sur les composants avec le *Framework* Molecule tels qu'ils sont utilisés au sein de l'équipe.

Au début du stage, il a été difficile de comprendre ce qu'est véritablement un composant. Il a fallu un temps de compréhension et d'adaptation car il s'agit d'un sujet complexe et tout nouveau pour moi.

Un questionnaire visible en annexe I a été élaboré afin d'interviewer les membres de l'équipe en tant que futurs utilisateurs de l'outil et de comprendre leur besoin et leurs attentes.

À travers cette interview, la démarche consistait :

- Dans un premier temps, la situation de la personne est identifiée à travers des questions sur sa journée-type, ses activités, ses missions, son environnement de travail, son matériel, ses langages utilisés et ses outils.

Par exemple, j'ai appris que mon tuteur de stage exerce différentes activités comme l'élaboration de documents, du maquettage, du développement logiciel avec un travail sur le design graphique et l'ergonomie. Il utilise des logiciels de dessin matriciels comme Photoshop ou des logiciels vidéos pour du montage.

- Dans un second temps, nous évoquons l'importance des composants au sein de son métier, par exemple, à quelle fréquence la personne utilise-t-elle ou crée-t-elle les composants.

Généralement, il s'agit d'une utilisation très fréquente quasiment quotidienne.

- Dans un troisième temps, nous procédons à l'analyse des points positifs et négatifs des composants. Les problèmes rencontrés, les avantages et leurs raisons sont partagés. Quelques solutions sont évoquées avec les éléments à améliorer.

L'architecture et la réutilisation sont les 2 points forts qui ressortent chez chaque membre de l'équipe.

C'était la première fois à titre personnel que je réalisais des interviews dans le cadre d'un projet. La principale difficulté était de se positionner en tant qu'intervieweur et de tenter de récolter les impressions des membres de l'équipe, même sur des détails, tout en gardant un discours objectif et neutre. En effet, il ne faut pas influencer la personne.

4.1.2 Analyse

Les résultats des interviews sont visibles en annexe II. Je les ai analysés à travers différents schémas et maquettes présentés ci-dessous.

À la suite des interviews réalisées avec chaque membre de l'équipe, l'étape suivante était de proposer une « *value proposition Canvas* » et un « *business proposition Canvas* » via le logiciel *Pencil*. Ce logiciel permet de réaliser des maquettes graphiques (voir annexe III et IV).

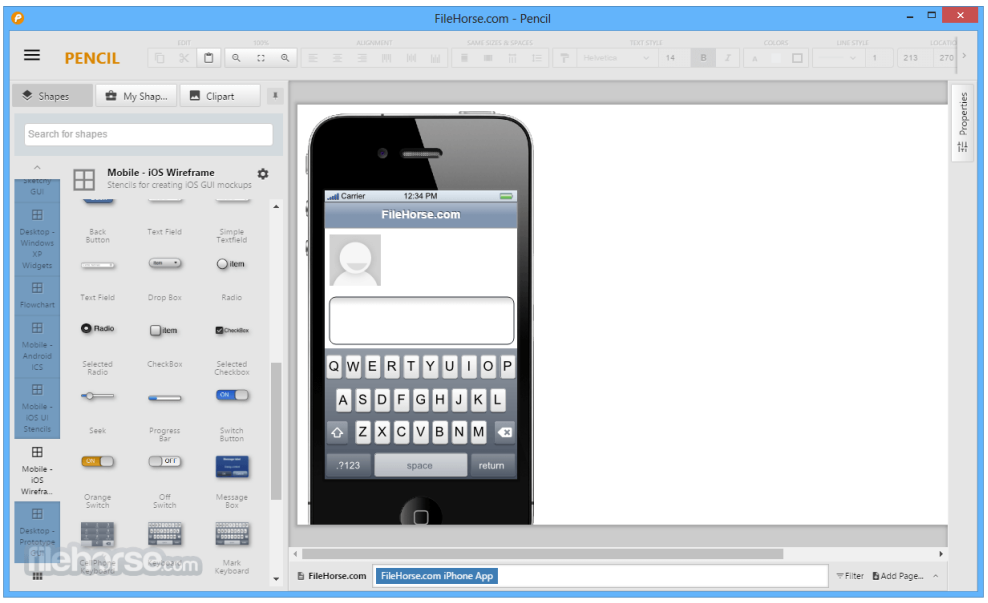

Figure 14 : Interface logiciel Pencil
Source : voir bibliographie

Il reprend les éléments évoqués par les utilisateurs en les associant avec l'outil qui est à développer. Ces deux réalisations m'ont permis de comprendre le besoin, cibler les problèmes et difficultés, proposer des idées pour trouver des solutions. L'un des objectifs était de pouvoir améliorer l'architecture des composants. Pour répondre à celui-ci, la détection d'anomalies avec leur résolution est une possibilité. Visualiser graphiquement un composant permet également de l'identifier plus rapidement et donc détecter ses anomalies plus rapidement aussi.

Par ailleurs, la procédure de création est aujourd'hui longue et répétitive ainsi l'objectif était de pouvoir réduire ce temps en automatisant certaines étapes. L'idée de créer un éditeur graphique est également prise en compte.

Puis vient la rédaction d'un document World liste les différentes fonctionnalités que pourraient implémenter l'outil (voir annexe V). Cette liste permet d'approfondir la « value proposition » en détaillant les tâches plus précisément. Les principales fonctionnalités attendues sont notamment la visualisation, la création, la modification, la recherche des composants et l'identification d'anomalies.

À partir de ce moment-là, un Modèle Conceptuel de Données en annexe VI (MCD) a été réalisé via le logiciel Yed. Il représente les données que possèdent un composant afin de connaitre précisément les informations qui seront importantes à prendre en compte lors du développement du logiciel. L'application Yed permet de créer des diagrammes, il s'agit d'un modeleur.

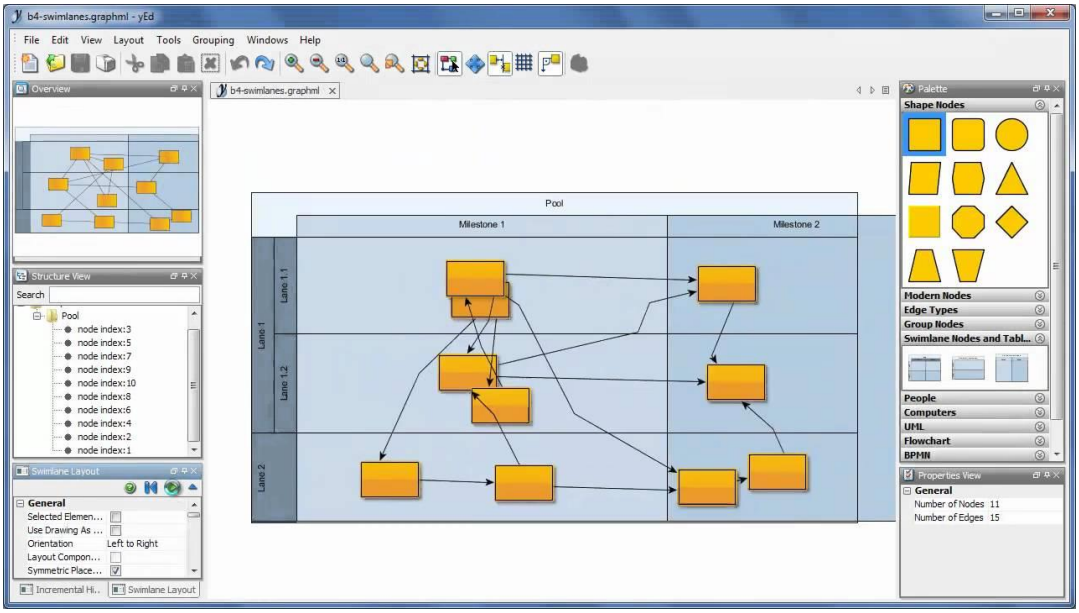

Figure 15 : Interface logiciel Yed
Source : voir bibliographie

À partir de l'analyse des informations recueillies, prioriser les fonctionnalités à développer est essentiel. Cette étape est fondamentale compte tenu de la durée du stage, elle permet d'identifier le niveau d'importance de chaque tâche, et elle me servira dans mes choix lors de la phase de développement.

Le tri a été effectué en fonction du nombre d'occurrences remonté par les interviews par rapport l'effectif total soit cinq au total. Cela a permis de concevoir deux graphiques sous forme de diagramme avec cette notion de priorité visible en pourcentage en figure 13 et 14.

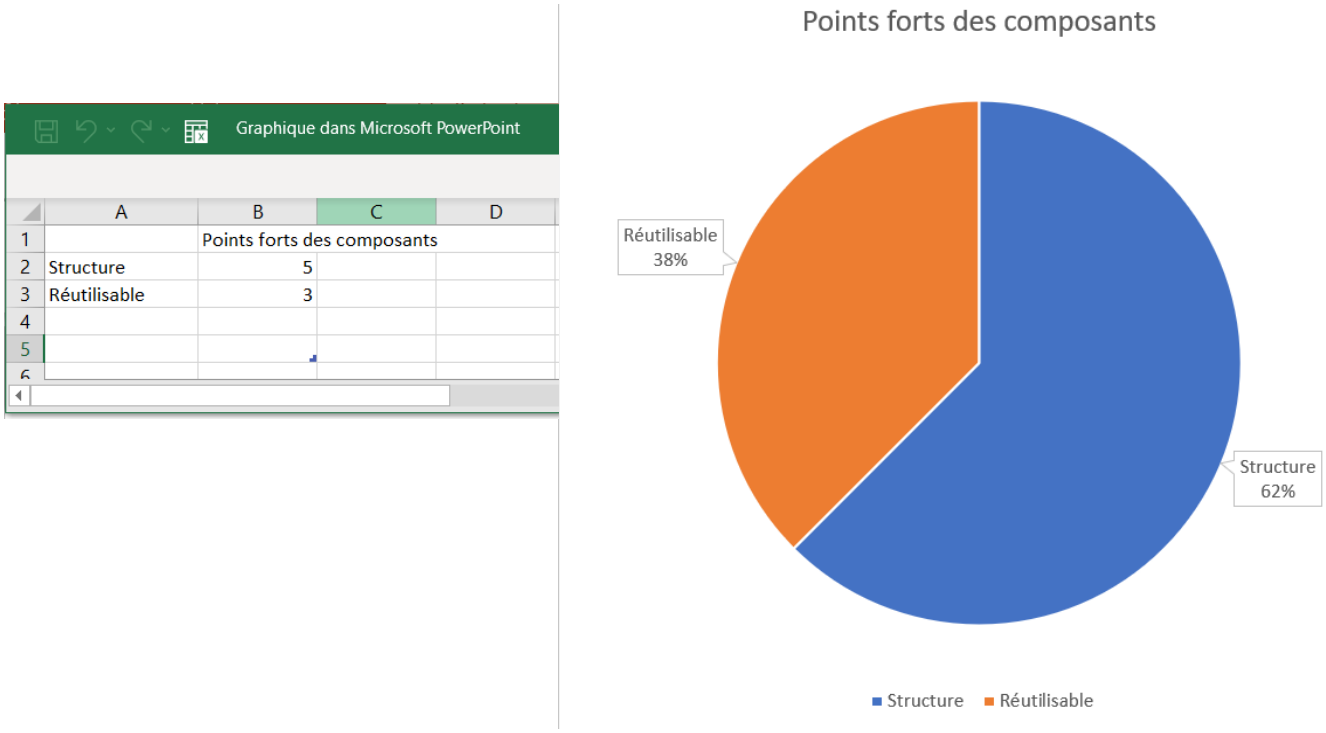

Figure 16 : Diagramme représentant les points forts des composants

Dans ce graphique, cinq personnes, soit la totalité de l'équipe, ont mentionné la structure. Puis, trois personnes ont parlé de la réutilisation. Ces deux aspects sont les plus importants, l'objectif va donc être, à travers mon projet, d'améliorer et de mettre en avant ces aspects.

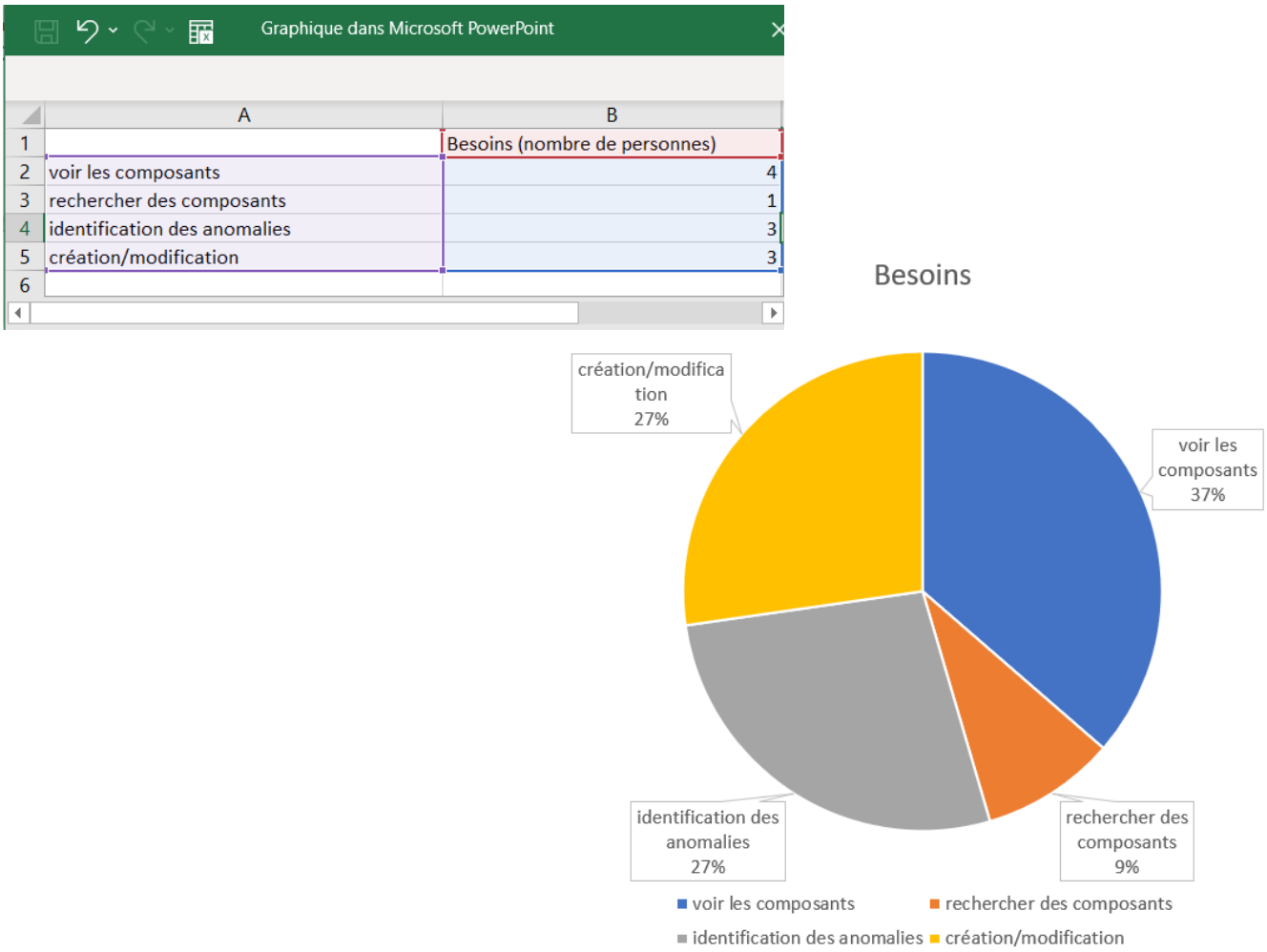

Figure 17 : Diagramme représentant le besoin des utilisateurs

Dans ce graphique, quatre personnes sur cinq ont évoqué le besoin d'un éditeur graphique permettant de visualiser le composant et ainsi identifier rapidement son contrat, son nom, ses liens avec les autres composants. Trois personnes ont parlé de la création du composant, qui pourrait se faire de manière graphique, ou bien optimiser et automatiser certaines étapes qui sont aujourd'hui trop répétitives et longues. De plus, cette aide à la création pourrait homogénéiser les pratiques. Trois personnes ont exprimé le besoin de l'identification des anomalies qui permettraient de réduire les problèmes d'architectures. Enfin, une personne a cité un outil de recherche qui permettrait de trouver plus rapidement un composant.

Au travers de ces analyses, la conclusion était que la visualisation, la création du composant et l'identification des anomalies sont les besoins les plus mis en avant par les utilisateurs.

Ces données ont permis de conclure que la visualisation graphique du composant était la première chose à faire avant de pouvoir par la suite l'enrichir avec différentes données et liaisons avec d'autres composants. Parallèlement au développement de cette fonctionnalité, un menu de monitoring (la surveillance) permettrait d'avoir des connaissances sur une instance d'un composant activée ou encore aiderait à savoir combien de composants sont activés à l'instant t. La création d'un composant sera dans un second temps à prendre en compte. Le principe est de reprendre l'élément visuel initial d'un composant et pouvoir le lier graphiquement avec d'autres composants comme l'utilisateur le souhaite.

4.1.3 Périmètre du projet

Cette analyse du besoin a permis aussi de mettre en lumière d'autres fonctionnalités qui peuvent susciter un intérêt mais à plus long terme. Par exemple, l'aspect *debug* pourrait être une aide précieuse à la suite de l'identification d'anomalies sur un composant. Malgré tout, l'intégration de cette fonctionnalité dans le projet actuel nécessite une réflexion d'un point de vue architecturale. En effet, ce module pourrait faire l'objet d'un outil à lui tout seul. Dans le cadre du projet actuel, le développement de cette partie a été jugé trop ambitieux selon le tuteur de stage.

4.2 Maquettage

Parallèlement à la phase d'idéation, quelques premières maquettes ont été commencées et sont régulièrement en évolution compte tenu des remarques des utilisateurs ou des nouvelles idées apportées par l'équipe lors de points de présentations.

Ces maquettes ont été réalisées à l'aide du logiciel *Pencil*.

Un exemple de l'une de ces maquettes concerne la visualisation des composants (voir annexe VII). Sur cette maquette, l'utilisateur a la possibilité de chercher un composant via une barre de sélection déroulante et d'afficher le composant graphiquement avec son contrat. On peut également voir les relations qu'il possède avec d'autres composants. L'annexe VIII

présente une évolution de cette maquette. Des modifications ont été apportées, par exemple, l'identification d'anomalies et le statut du composant ont été ajoutées via un code couleur.

Ces maquettes sont importantes pour définir le projet, donner un aperçu du rendu visuel de l'outil et emporter l'adhésion de l'équipe. La définition du projet était en cours de construction ainsi des étapes de discussions et d'échange avec l'équipe permettent de converger vers les bonnes solutions collectivement et ont pu faire avancer le projet.

Selon moi, cette étape est importante au sein du projet. Il s'agit de retranscrire le besoin visuellement. Aussi, les maquettes servent d'intermédiaires entre la compréhension du besoin et la phase de développement.

4.3 Prototypage et réalisation

À la suite de la phase de maquettage, il a fallu passer au codage de l'application dans l'environnement Pharo.

À ce moment, débute la prise de connaissance sur la syntaxe Pharo ainsi que les modules intégrés qui sont Roassal3 et Spec2. L'assimilation s'est faite via des exemples disponibles sur GitHub que j'analysais, je reproduisais et modifiais dans mon image Pharo. J'ai aussi utilisé les outils disponibles dans Pharo afin de déboguer[13] pas à pas le code que j'exécutais notamment grâce au débogueur, au *Playground* et la console (*Transcript*).

J'ai commencé le développement en réalisant un point d'entrée via un menu contextuel pour ouvrir une fenêtre avec des informations sur le composant sélectionné (voir annexe IX). Roassal3 n'est pas utilisé pour ce premier pas dans l'environnement Pharo.

Le but était de développer quelque chose de très simple et fonctionnel et l'enrichir progressivement enrichir progressivement afin d'obtenir un résultat visible dès le début de la phase de développement en raison de la courte durée du stage. L'avantage est que mon tuteur a accès au code source (via *GitHub*) et peut tester l'application. Il peut me faire rapidement des retours si des améliorations ou des bugs sont à corriger.

[13] **Déboguer** : Enlever et éliminer les erreurs d'un logiciel.

Pour réaliser ce menu et cette fenêtre, au niveau du codage, la première difficulté a été de comprendre comment fonctionne un composant, ou les informations se situent et comment y accéder. J'ai dû faire un travail de recherche et de compréhension en amont. Ces actions ont été nécessaires, elles ont pris plus de temps que je l'imaginais.

Concernant l'outil visuel, cela a commencé par le développement d'une fenêtre permettant de visualiser graphiquement l'ensemble des composants sous forme de nuage de points. Chaque composant est représenté par son icône. C'est une vue globale des composants (voir annexe X).

C'est à ce moment précis que j'ai commencé à utiliser Roassal3. J'ai d'abord dû intégrer Roassal3 dans une fenêtre Spec2 avant de me lancer dans le développement de la visualisation de l'ensemble des composants. Le but était d'obtenir une fenêtre avec un dessin graphique à l'intérieur.

Parmi les composants, la sélection individuelle est possible. L'utilisateur arrive alors sur sa vue détaillée (voir annexe XI). Cette vue est en quelque sorte un « zoom » sur le composant. Ses informations (ses interfaces, son contrat, son *Type*…) sont récupérées et retranscrites graphiquement.

Cette vue m'a prise un peu de temps car j'ai dû récupérer les bonnes informations pour chaque composant et également assembler l'ensemble des petits éléments graphiques pour afficher le composant complet.

Par la suite, une nouvelle vue est réalisée reprenant la vue détaillée d'un composant dont le but est différent. En effet, cette fenêtre permet de visualiser les instances d'un composant (voir annexe XII) et permet également d'avoir plusieurs informations comme le nombre d'instances, leurs noms, celles qui sont activées ou arrêtées.

J'y ai ajouté des interactions avec l'utilisateur. Il peut aussi feuilleter les différentes instances par l'intermédiaire de boutons *Next* et *Previous*.

Enfin, une autre vue permet d'éditer graphiquement un composant. Cette vue a pour objectif de créer un composant au travers de sa modélisation (voir annexe XIII).

Cette vue a été réalisé à la fin. Je n'ai pas pu la finir par manque de temps. Il s'agit d'un début d'éditeur graphique. Par exemple, l'utilisateur peut créer des composants mais il ne peut pas l'éditer, c'est-à-dire qu'il ne peut pas changer son contrat ni son *Type*.

Toutes ces vues ont bien entendu évolué durant les semaines de développement, elles se sont enrichies en ajoutant certains aspects visuels et certaines fonctionnalités comme la possibilité de pouvoir rechercher un composant via un menu déroulant. Certaines vues sont statiques et d'autres dynamiques, elles sont accompagnées par des interactions avec l'utilisateur comme des raccourcis claviers et des informations actualisées.

Tout au long du projet, une documentation a été rédigé afin d'effectuer des points d'étapes, de commenter mon code et de réaliser des tests unitaires.

Enfin, ce projet a permis d'expérimenter la plateforme collaborative GitHub et de contribuer à la collectivité sous forme de publications de rapports de bugs concernant l'ensemble des exemples disponibles concernant Roassal3, Spec2 et Pharo. J'ai contribué à l'évolution et au développement de Spec2 et Roassal3 en remarquant certains points à améliorer ou d'autres à créer.

4.4 Réflexion

Je trouve que les différentes phases du projet sont pertinentes. J'ai appris à travers ce projet à récolter le besoin de l'utilisateur en premier et qu'il s'agit de l'élément fondamental du projet.

5 Bilan

5.1 Bilan pour l'entreprise

Au sein de Thales, pour l'équipe UX & prototypage, l'enjeu de ce stage était d'effectuer des travaux de tests sur des technologies qu'elle souhaite pousser à l'avenir. Le périmètre du projet était clair et adapté pour un stage de 10 semaines. Leur volonté était d'évaluer le temps de prise en main et d'appropriation des outils internes de l'entreprise sur des profils de développeurs externes.

L'objectif de l'entreprise est également de migrer progressivement de Smock 1 vers Smock 2. Le projet de stage a été motivée par l'absence de communauté de développeurs du côté de VisualWorks contrairement à Pharo où la communauté est très active. C'est un aspect qui intéresse beaucoup les membres de l'équipe. Ce stage contribue également à cet objectif.

Ce projet permet aux développeurs de faire avancer leur atelier de prototypage IHM en y ajoutant des outils pour augmenter la productivité du développement logiciel. Il sera utilisé lors du prochain prototype développé par l'entreprise.

Les remontées de l'équipe lors des points présentation sont encourageantes sur le fait qu'il sera utilisé dans le futur. Dans ce sens, on peut estimer que l'objectif est rempli. L'objectif initialement fixé par l'équipe et moi-même a été accompli. L'outil va vraiment aider l'équipe à gagner en efficacité.

Les cinq membres de l'équipe ont beaucoup d'idées pour enrichir l'outil et y ajouter de nouvelles capacités. Cet outil sera intégré directement dans Molecule en open-source et par extension à Smock 2 (l'atelier de prototypage IHM de l'équipe).

Concernant la suite du projet, l'outil sera industrialisé et il faudra le rendre plus robuste. Cette suite ne fait pas partie du périmètre du stage.

5.2 Bilan humain

L'environnement de travail au sein d'une grande entreprise internationale a été une réelle découverte pour moi.

Je suis très contente d'avoir effectuée ce stage au sein de l'équipe UX & prototypage de Thales. J'ai pu découvrir le rôle de chacun et comment le groupe collabore sur les projets. Notamment, j'ai pu relever qu'une bonne communication est un aspect majeur à la réussite de leur mission. De nombreux outils sont à leur disposition, des outils techniques (mail, tchat…) comme des méthodes (points hebdomadaires…).

J'ai également eu la chance de travailler en présentiel, ce qui m'a permis de me plonger dans le monde professionnel. J'alternais des temps de télétravail suivant l'organisation de mon tuteur entre un à deux jours par semaine.

Je me suis rendu compte que plusieurs membres de l'équipe télétravaillaient régulièrement quelques jours dans la semaine. Le fait d'être ingénieur logiciel permet de travailler à distance puisque le matériel principal utilisé reste l'ordinateur.

Durant mon stage, j'ai eu l'occasion d'assister à plusieurs réunions :

- La première avait pour but de réaliser un point sur les objectifs de l'équipe UX. Lors de chaque objectif évoqué, chaque membre de l'équipe attribuait une note entre un et cinq du degré de confiance qu'il avait concernant la faisabilité de cet objectif ;
- La deuxième avait pour objectif de réaliser un point d'avancement sur les activités effectuées par les membres de l'équipe ;
- La troisième avait pour objectif de réunir l'ensemble du département Discipline Logiciel Industrielle afin de présenter les actualités du département comme les projets actuels et à venir.

Durant mon stage, j'ai également organisé une réunion avec deux membres de l'équipe. L'objectif était de leur présenter plusieurs maquettes afin d'avoir leurs premiers retours et impressions.

J'ai pu aussi présenter mon projet à l'ensemble de l'équipe à la fin de mon stage.

J'ai appris et découvert le métier d'UX & UI designer et cette découverte me conforte sur mes choix d'orientation futurs. J'envisage d'ailleurs de poursuivre mes études dans ce domaine.

Enfin, l'ambiance au sein de l'équipe est très bonne. Ils ont su m'intégrer et répondre à mes questions afin de faire évoluer mon projet.

5.3 Bilan pédagogique

À travers ce stage, j'ai pu enrichir mes connaissances et compétences en informatique. En effet, j'ai découvert les méthodes de projet UX tel que le *Design Thinking*.

L'apprentissage d'un nouveau langage (Pharo) fait également partie des plus-values de ce projet. Ce langage possède des caractéristiques différentes des langages de programmation appris à l'IUT tels que les langages Java et Python. Par exemple, le typage des variables et dynamique contrairement à un typage statique pour les langages comme Java et Python. Ce langage Pharo retrouve quand même des propriétés similaires aux langages objet comme Java tels que la notion d'héritage et la gestion de la mémoire qui est dynamique. Ces notions se sont donc renforcées durant mon stage.

Il m'a fallu un peu de temps pour comprendre les principes et le fonctionnement de ce langage. Une fois ce temps passé, j'ai codé l'outil sans trop de blocage. Je suis assez satisfaite de mon travail réalisé même si je souhaitais faire encore plus. Concernant le langage, j'ai bien apprécié coder avec, je trouve que la syntaxe est simple à comprendre, les automatismes arrivent rapidement.

J'ai aussi appris à utiliser un *Framework* nommé Spec2 et une bibliothèque graphique appelée Roassal3. Pour ma part, j'ai trouvé que Roassal3 était plus facile à prendre en main que Spec2. En effet, grâce à Roassal3, selon moi, il était plus facile d'arriver plus rapidement à un résultat visuel. Par exemple, en quelques lignes de code, il est facile de dessiner et personnaliser une forme géométrique dans une fenêtre. Tandis que Spec2, selon moi, il était plus difficile d'arriver à un résultat satisfaisant. J'ai trouvé que les possibilités d'afficher des éléments et de pouvoir les personnaliser sont plus restreintes que Roassal3. Par exemple, je souhaitais

afficher une barre de recherche avec une liste déroulante qui s'actualisait dynamiquement en fonction de la saisie de la recherche. Il se trouve que l'élément représentant la barre de recherche et l'élément représentant la liste déroulante sont deux objets différents qui sont difficiles à coupler.

De plus, Roassal3 propose davantage d'exemples que Spec2 ce qui permet d'avoir plus de choix pour coder.

Conclusion

Ce stage fut pour moi l'opportunité de travailler dans une entreprise du secteur industriel pour la première fois. Je pense qu'il a été très bénéfique pour appréhender le monde du travail. Thales est un grand groupe international que j'ai pu apprécier au travers des nombreux projets développer au sein de cette structure et du nombre de salariés. J'ai d'ailleurs eu l'occasion de découvrir et de tester un simulateur de vol.

L'équipe qui m'a accueilli compte cinq personnes dans un département d'une cinquantaine de personnes. J'ai pu me rendre compte que la communication et les échanges entre les équipes doivent être réguliers afin de faire avancer l'ensemble des projets et mettre en avant les valeurs de la société.

Du coté de mes compétences techniques, ce stage m'a apporté de nouvelles connaissances et compétences informatiques.

Ce stage m'a permis de prendre de la hauteur et d'aborder un projet sous un nouvel angle en partant du besoin de l'utilisateur plutôt que sur un descriptif de fonctionnalités déjà existant.

L'outil développé durant le stage est à présent fonctionnel, cependant, je n'ai pas pu faire l'intégralité des fonctionnalités souhaitées par les utilisateurs. Des évolutions et améliorations sont donc fortement envisageables.

Tables des illustrations

Bibliographie et sitographie

1. Bibliographie

- LABORDE Pierre, COSTIOU Steven, PLANTEC Alain, Le PORS Eric, « 2020 - 15 years of reuse experience in evolutionary prototyping for the defense industry », Brest, Thales DMS France, INRIA, UBO, 2020, lien : https://hal.inria.fr/hal-02966691/ ;

- LABORDE Pierre, COSTIOU Steven, PLANTEC Alain, Le PORS Eric, « Molecule: live prototyping with component-oriented programming », Brest, Thales DMS France, INRIA, UBO, 2020, lien : https://hal.inria.fr/hal-02966704/.

2. Sitographie

« *Pencil* », lien du site officiel : Home - Pencil Project (evolus.vn)

« Yed », lien du site officiel : https://www.yworks.com/products/yed

« Molecule », lien du GitHub : GitHub - OpenSmock/Molecule: Molecule is a Pharo component framework.

« SmallTalk », lien *Wikipédia* : https://fr.wikipedia.org/wiki/Smalltalk

« Pharo », lien du site officiel : Pharo - Welcome to Pharo!

« Pharo », lien du GitHub : GitHub - pharo-project/pharo: Pharo is a dynamic reflective pure object-oriented language supporting live programming inspired by Smalltalk.

« Pharo », lien du GitHub pour la documentation : https://github.com/pharo-open-documentation/pharo-wiki

« Pharo », lien *Wikipédia* : https://fr.wikipedia.org/wiki/Pharo

« Pharo », lien discord (communauté) : https://discord.gg/QewZMZa

« Roassal », lien du site officiel : http://agilevisualization.com/

« Roassal », lien du GitHub : GitHub - ObjectProfile/Roassal3: The Roassal Visualization Engine

« Spec », lien du GitHub : https://github.com/pharo-spec/Spec

« Inria Learning Lab », MOOC pour apprendre pharo, lien de la chaîne Youtube : https://www.youtube.com/channel/UClr-bzIv10z27ONlSjmosYg/featured

Tables des annexes

Annexe I : Questionnaire pour les interviews

Questionnaire

- ➔ Question n°1 : Quelle est sa journée-type ? son environnement ? Quelles sont ses outils et ses langages de programmation ?
- ➔ Question n°2 : A quelle fréquence crées-tu et utilises-tu des composants ? Sur Smock 1, Smock 2 ? A quoi cela va te servir ?
- ➔ Question n°3 : Quels sont, selon toi, les points forts des composants ? Pourquoi ?
- ➔ Question n°4 : Comment pouvoir améliorer ces points forts ?
- ➔ Question n°5 : En revanche, quels sont les points faibles des composants ? Comment pouvoir diminuer et résoudre ces points faibles ?
- ➔ Question n°6 : Lors de la création, modification, qu'est-ce qui te pose problèmes et pourquoi ?
- ➔ Question n°7 : Comment faire pour réduire ces problèmes ? As-tu des idées ou des solutions ?
- ➔ Question n°8 : Au contraire, qu'est-ce que tu trouves pratique, facile à utiliser ?
- ➔ Question n°9 : Qu'est-ce qui pourrait être amélioré et pourquoi ?
- ➔ Question n°10 : Aimes-tu créer des composants ? Pourquoi ?
- ➔ Question n°11 : Lors de la création, quels sont les champs importants à connaitre ? Les données à fournir ? Son contrat, son *Type* ?

Annexe II : Résultats des interviews

Résultats

Première personne :

Réponse à la question n°1 :

- Maquettage => *WireFraming* / *Sketching* => capture du besoin
- Configuration de PC Linux pour démonstration
- Travail en équipe
- Déplacements occasionnels pour rencontrer les clients, sinon principalement au poste
- Langages : Java, SmallTalk, VisualWorks, Pharo
- Outils : PicPick Pencil et Yed pour les maquettes

Réponse à la question n°2 :

- Quotidiennement, sur Smock 1, pour le prototypage

Réponse à la question n°3 :

- Points forts : la structure du composant avec son contrat => ses interfaces (services et events)

Réponse à la question n°6 :

- Processus long et répétitif => appliquer le *DefinedComponent* à chaque fois
- Lors de la modification c'est plus problématique que lors de la création
- Le processus d'abonnement est également un peu pénible
- Cela engendre des oublis le fait que ce soit long et répétitif => manuel pas automatique

Réponse à la question n°5 :

- Problème d'architecture : dépendance entre les composants : parfois un composant doit être activé avant un autre composant
- Confusions entre les méthodes privées et publiques de temps en temps

Réponses à la question n°7 :

- Système permettant de vérifier les méthodes.

Réponse à la question n°8 :

- Automatiser certaines étapes
- Utilise un générateur qui définit et crée les services fournis

Réponse à la question n°11 :

- Les interfaces et le *Type* du Composant (pas de trait dans Smock 1)

Deuxième personne :

Réponse à la question n°1 :

- UX lead => porter les valeurs de l'UX
- Collaborer avec les autres / Travail en équipe
- Chercher des projets/sujets UX
- Pilotage et expertise UX => développement logiciel (double casquette)
- Comprendre les autres, être empatique
- Déplacements occasionnels pour rencontrer les clients, sinon au poste
- Langages : Java, SmallTalk, VisualWorks, Pharo, C
- Outils : PicPick Pencil et Yed pour les maquettes

Réponse à la question 2 :

- Création régulière de composant

Réponse à la question n°3 :

- Points forts : Structure et réutilisation des composants
- Pouvoir utiliser un composant dans un autre => rapide

Réponse à la question n°5 :

- Point faible : il faut avoir l'esprit structuré, il faut toujours confronter les composants avec une autre personne.

Réponse à la question n°10 :

- Aime bien faire les composants cœurs car plus importants que les composants feuilles

Troisième personne :

Réponse à la question n°1 :

- Maquettage
- Collaborer avec les autres / Travail en équipe
- Support expertise
- Industrialisation
- Reste au poste régulièrement
- Langages : SmallTalk, VisualWorks

Réponse à la question 2 :

- Utilisation régulière => journalière, Smock 1 (VisualWorks)

Réponse à la question n°3 :

- Points forts : la réutilisation du composant et la structure => pouvoir exposer facilement le fonctionnel d'une sous-partie

Réponse à la question n°5 :

- Ils peuvent être dépendants du logiciel, on peut avoir un composant par fonctionnalité

Réponse question n°6 :

- La création du composant peut prendre du temps en fonction de la taille du composant

Réponse à la question 9 :

- La recherche des composants => outil => ouvre un éditeur pour trouver les composants

Réponse à la question 11 :

- Définir le nom, les services et les messages proposés par les services.

Quatrième personne :

Réponse à la question n°1 :

- Maquettage / Prototypage
- Collaborer avec les autres / Travail en équipe
- Effectue des points d'avancement
- Modèle de conception
- Langages : SmallTalk, Pharo

Réponse à la question 2 :

- Utilisation tous les jours, Smock 2

Réponse à la question n°3 :

- Garantie une bonne architecture => pratique, on ne peut plus sans passer.

Réponse à la question n°4 :

- Problèmes d'architectures et des problèmes techniques sont présents. Parmi eux, les composants, les traits sont mal tirés, les liens sont mal faits, les moyens pour accéder à un composant peuvent être détournés.

Réponses à la question n°6 :

- Processus de création pénible sur SCM => plus rapide sur Molecule avec les *Traits*.

Réponses à la question n°7 :

- Système permettant de voir quel ou quel composant tourne.
- Editeur graphique permettant de voir le contrat, le nom, les connexions du composant.

Réponse à la question n°11 :

- *Type*, services, évènements et paramètres

Réponse à la question 9 :

- Déploiement / Instanciation / Activation => long => avoir un *Launcher* permettant d'activer automatiquement un groupe de composant suivant des délais d'attentes.

Cinquième personne :

Réponse à la question n°1 :

- Développement logiciel => atelier de prototypage
- Collaborer avec les autres / Travail en équipe
- Design graphique / ergonome => IHM => Décrire l'ergonomie d'une interface
- Maquettage => croquis
- Groupe de travail avec les clients => 1 à 2 fois par mois => formaliser le besoin, réunion, discussion, atelier
- Equipement : Clavier, Souris, Ecran
- Langages : SmallTalk, Pharo, VisualWorks

Réponse à la questionn°2 :

- Utilisation tous les jours, Smock 2

Réponse à la question n°3 :

- Garantie une bonne architecture => code bien rangé, propre => facile pour naviguer => on connait le rôle du composant, ce dont il a besoin, ce qu'il fait => pas d'ambigüité
- Smock => tous se connecte tout seul => riche et simple
- Réutilisation => tous les composants sont réutilisables par nature

Réponse question n°6 :

- La création du composant => rien est pratique

Réponse question n°7 :

- *RunTime* avec l'état / relation avec les instances

Réponse question n°9 :

- On a du mal à voir les composants qui existent déjà => ajouter au composant des tags/commentaires pour les identifier
- Contrat d'un composant => le même pour tous => outil d'aide à la décision pour savoir si le développeur doit réutiliser ou pas
- Cycle de vie important pour le développeur => l'orienter pour qu'il n'oublie pas

Annexe III : The *Value Proposition Canvas*

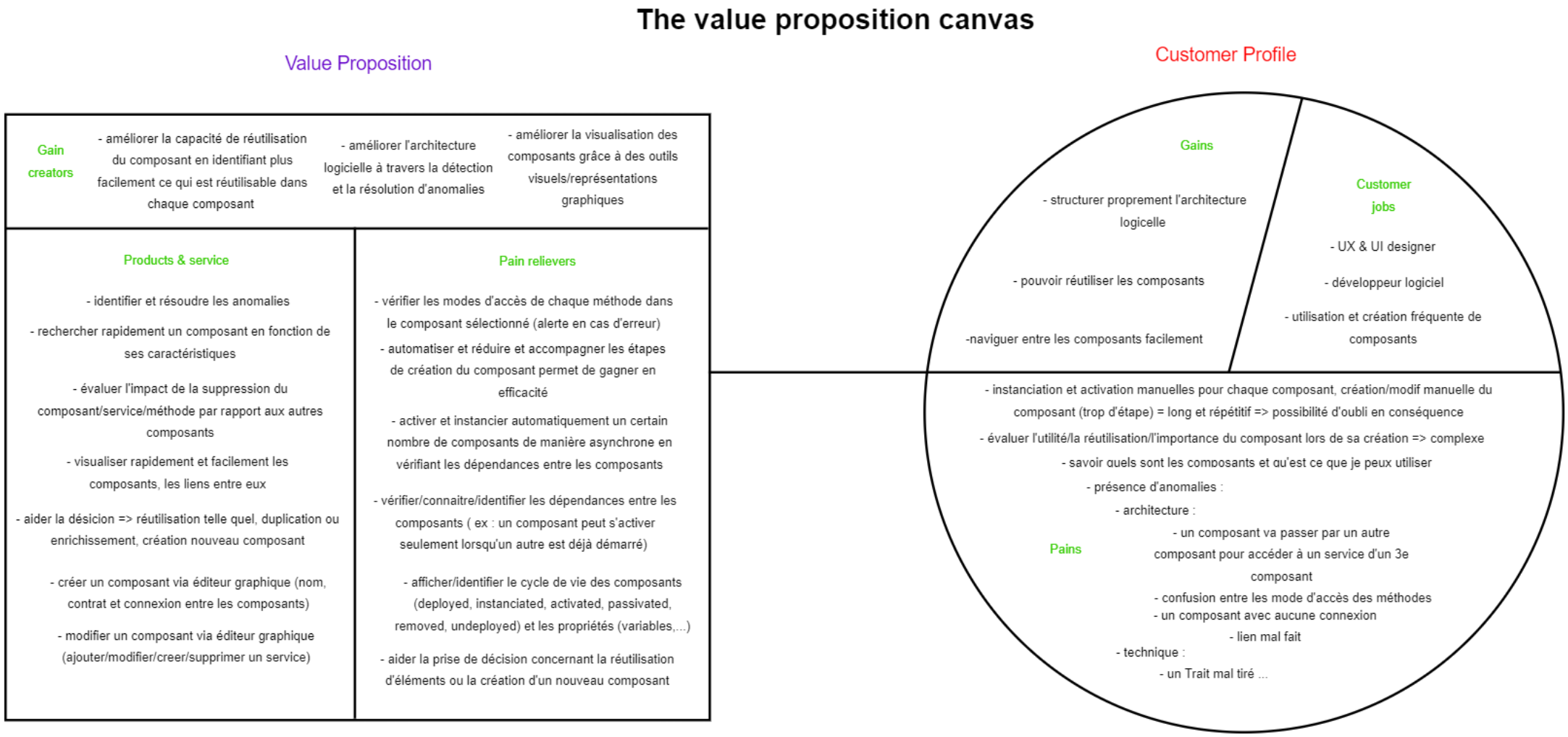

Figure 18 : The Value Proposition Canvas

Annexe IV : The *Business Model Canvas*

The business model canvas

<table>
<tr><th>Key Partnerships</th><th>Key Activities</th><th>Value Propositions</th><th>Customer Relationships</th><th>Customer Segments</th></tr>
<tr><td rowspan="3">- équipe UX de DLI
- sous-traitant équipe UX (main d'oeuvre)

- communauté open- source SmallTalk

- partenaires académiques : UBO et INRIA</td><td>- développement IHM logiciel
- prototypage IHM logiciel</td><td rowspan="3">Cf : Value Proposition</td><td>- équipe UX de DLI
- développeurs logiciels pharo
- sous-traitant équipe UX (main d'oeuvre)</td><td rowspan="3">Cf : Costumer Profile</td></tr>
<tr><td>Key Ressources</td><td>Channels</td></tr>
<tr><td>- Humains : membres de l'équipe
- Matériels : matériels informatique (clavier, écran, souris, ordinateur)
- Documentation : ressources bibliographiques</td><td>- contact par mail, téléphone, visioconférence
- fonctionnement open-source : GitHub, GitLab
- partenaires académiques : conférence, open source</td></tr>
<tr><td colspan="3">Cost Structure</td><td colspan="2">Revenue Streams</td></tr>
<tr><td colspan="3">- poste de travail pour Molecule</td><td>- produit + robuste</td><td>- faire évoluer le produit, ajouter des fonctionnalités</td></tr>
</table>

Figure 19 : The Business Model Canvas

Annexe V : Liste des fonctionnalités de l'outil

Liste des capacités/fonctionnalités de l'outil :

- Améliorer l'architecture :
    - Détection d'anomalies (système de vérification) :
        - Technique : les *Traits* mal fait/mal tiré
        - Architecture :
            - Un composant va passer par un autre composant pour accéder à un service du 3e composant (les moyens d'accès sont détournés)
            - Un composant qui ne possède aucune connexion
            - Confusion entre les modes d'accès des méthodes (privée, dans un service…)
    - Résolution de ces anomalies (savoir quelles sont les limites de mon outil)

- Connaitre/Savoir/Identifier les composants qui doivent être instanciés avant d'autres composants (dépendances entre les composants)

- Automatiser l'instanciation, l'activation, le déploiement des composants
    - Regrouper les composants et les instancier/activer suivants différents délais en respectant les dépendances

- Créer un composant :
    - Automatiser plusieurs étapes (les copier-coller, appliquer le *definedComponent* à chaque fois => valable que Smock 1)
    - Editeur graphique et/ou formulaire de saisi
        - Remplir le nom, le contrat, les connexions => composant créé

- Supprimer un composant ou un service/évènement/paramètre ou une méthode d'un composant :
    - Evaluer l'impact de la suppression par rapport aux autres composants
    - Avoir la possibilité de conserver/stocker le service ou la méthode afin de pouvoir les réutiliser derrière par exemple

- Modifier un composant :
    - Editeur graphique (comme à la création) et/ou formulaire de saisi
        - Permet d'ajouter, modifier, supprimer des évènements, services, paramètres et leurs méthodes

- Dupliquer un composant :

    - Copier-coller un composant et le modifier par la suite
    - Connaitre/Identifier rapidement à travers le contrat d'un composant ce qui est réutilisable ou non
    - Aider le développeur à prendre sa décision s'il doit créer un nouveau composant ou réutiliser des éléments d'un composant existant

- Rechercher un composant :
    - Recherche par nom/projet/catégorie
    - Ajout d'un système de référencement/documentation/tags/commentaire permettant d'identifier facilement les composants

- Voir les composants :
    - Visualiser facilement et rapidement le composant, son contrat, ses relations avec les autres composants à l'aide de schémas/graphiques

Annexe VI : MCD du composant

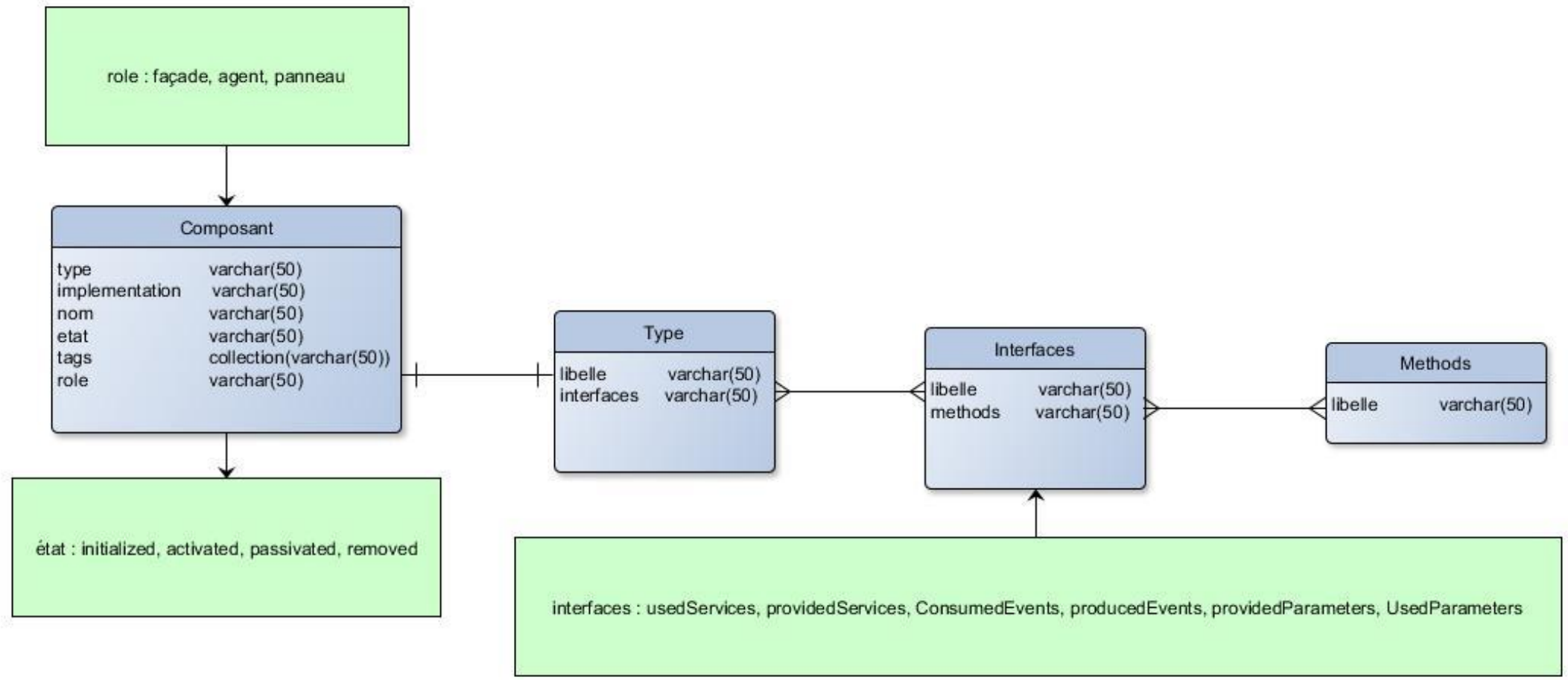

Figure 20 : MCD des composants

Annexe VII : Première maquette de l'outil

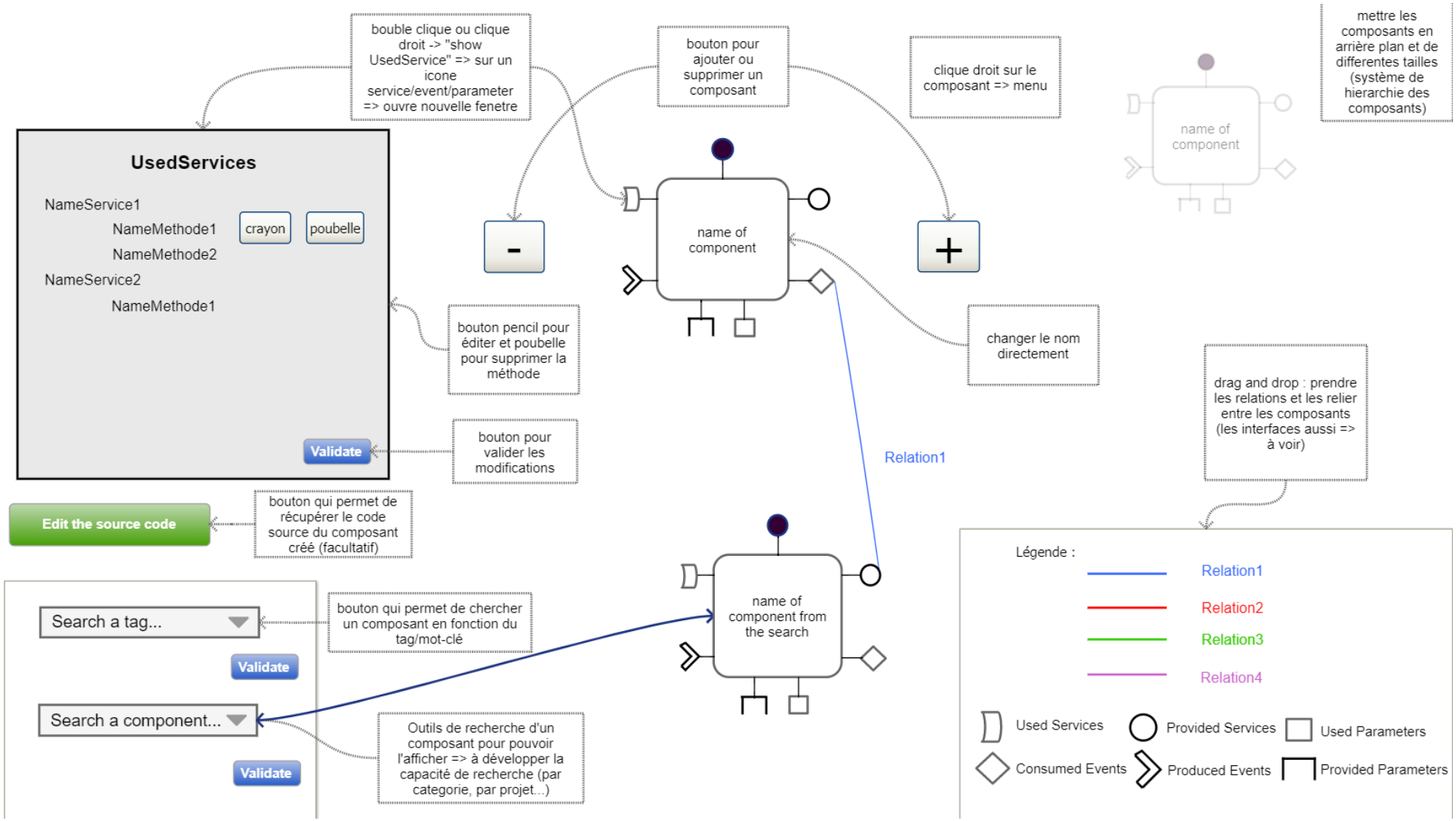

Figure 21 : Première maquette de l'outil

Annexe VIII : Deuxième maquette (évolution de la première maquette)

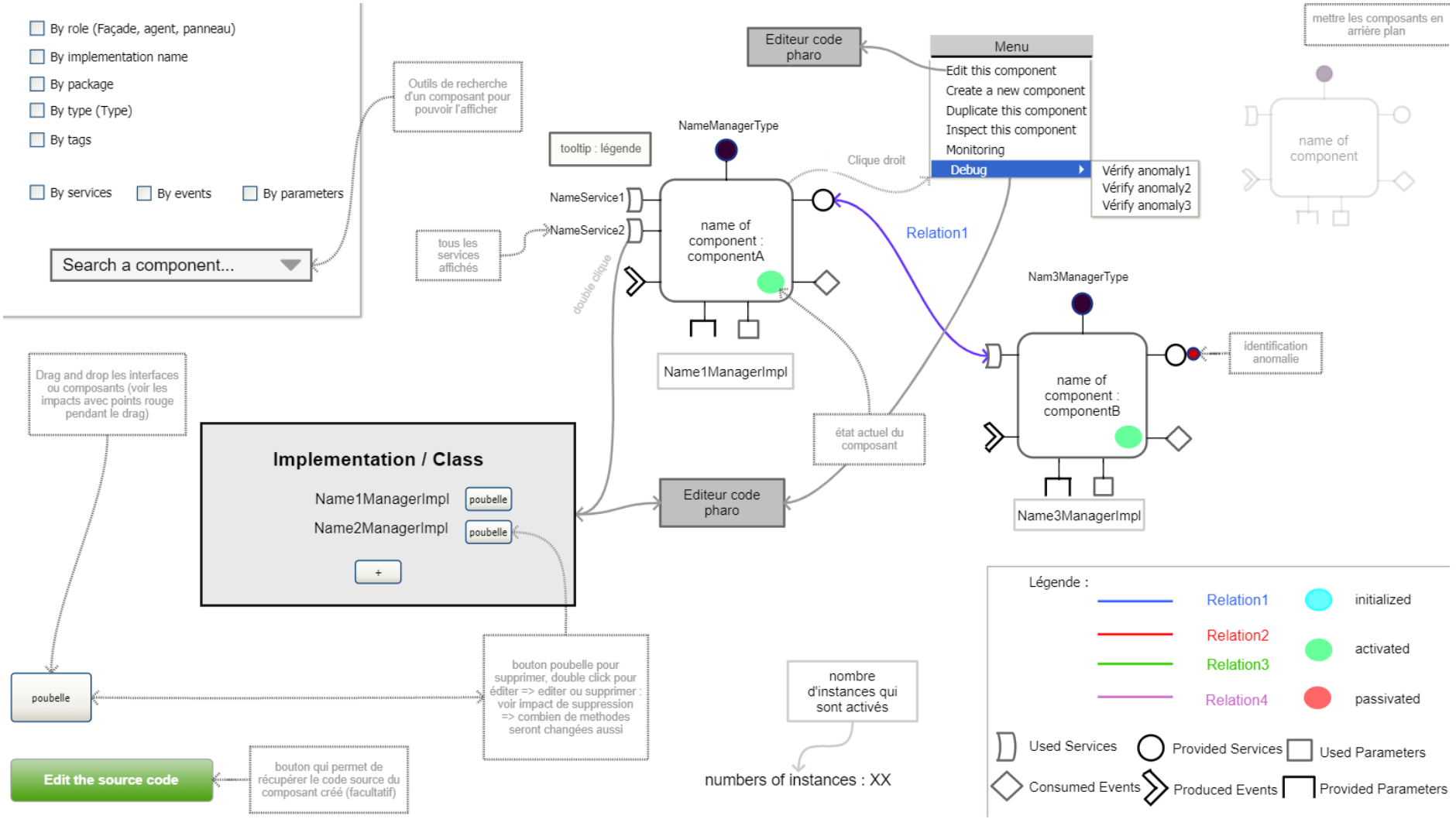

Figure 22 : Deuxième maquette des composants

Annexe IX : Fenêtre contenant les informations sur le composant

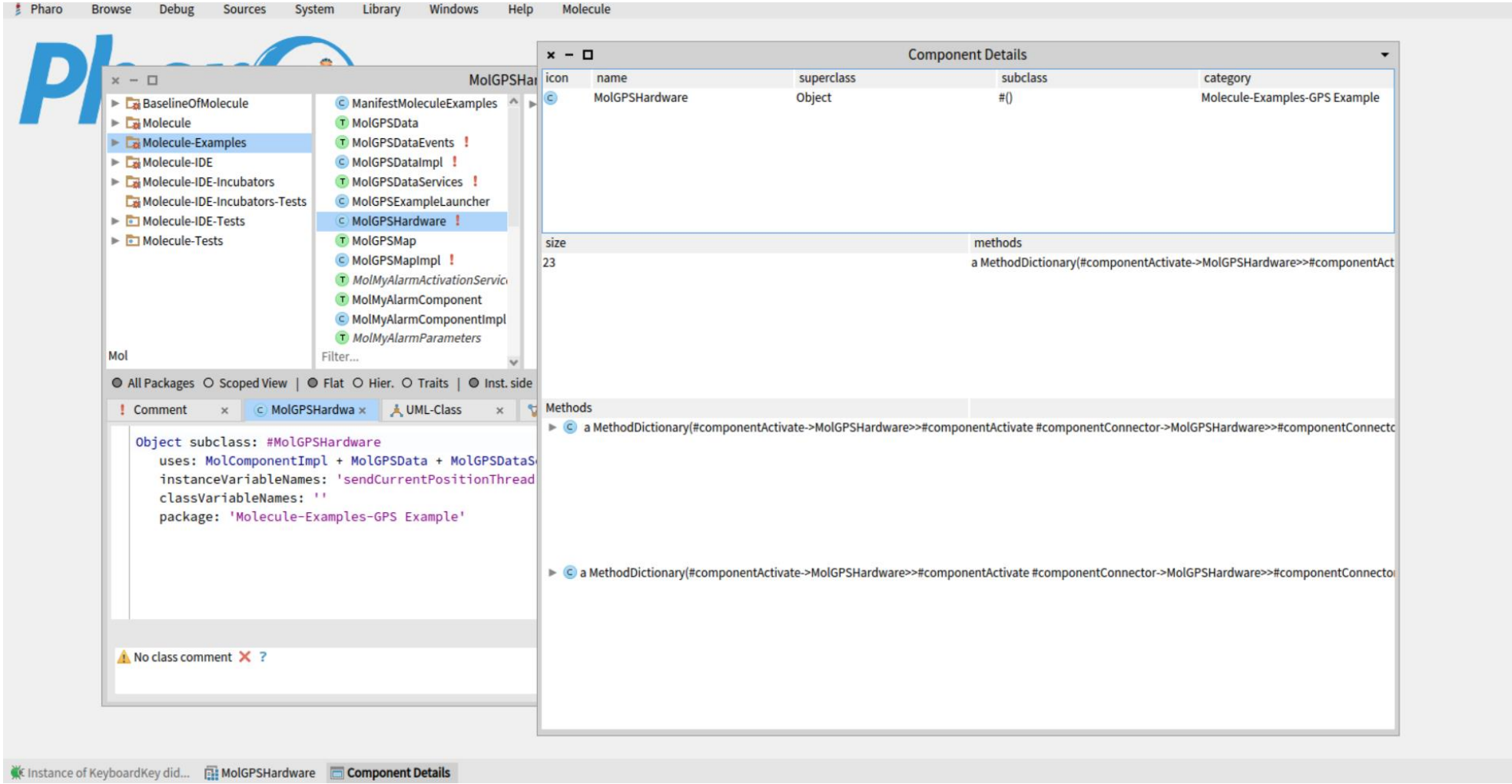

Figure 23 : Capture d'écran de la fenêtre contenant les informations sur le composant sur Pharo

Annexe X : Fenêtre correspondant à la visualisation du nuage de points des composants

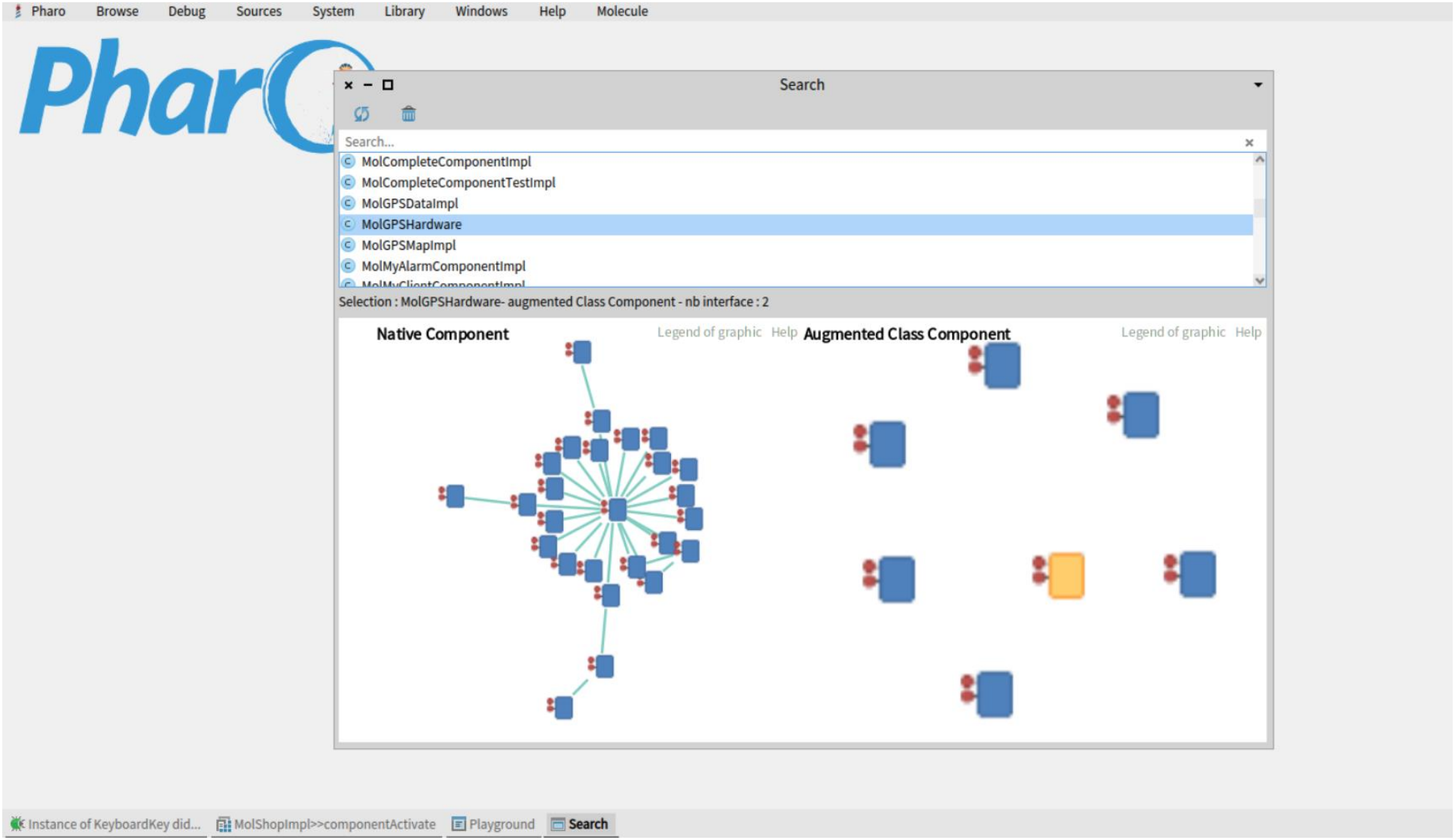

Figure 24 : Capture d'écran de la fenêtre du nuage de points des composants sur Pharo

Annexe XI : Fenêtre correspondant à la visualisation détaillée d'un composant

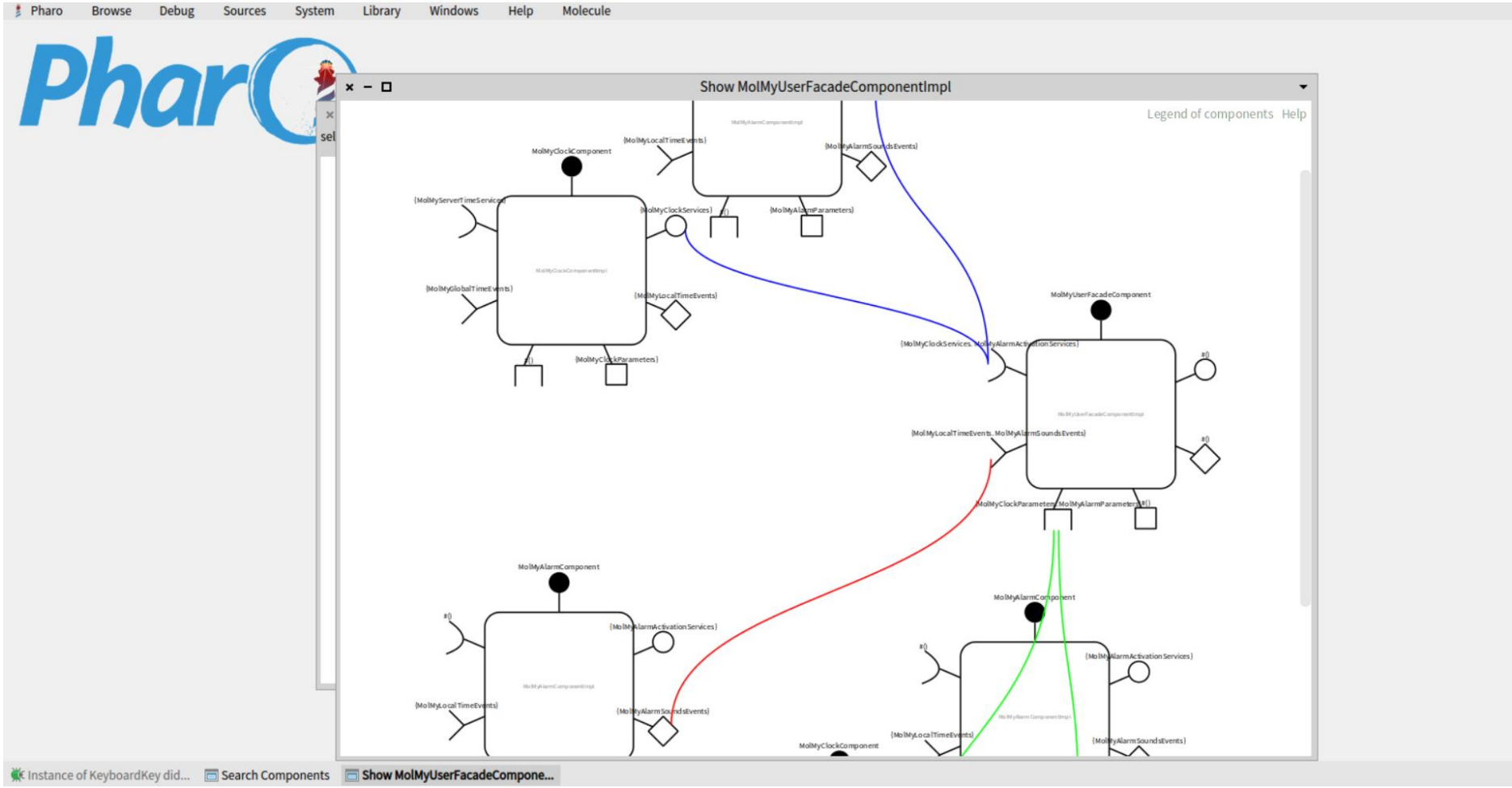

Figure 25 : Capture d'écran de la fenêtre correspondant à la vue détaillée d'un composant sur Pharo

Annexe XII : Fenêtre correspondant à la visualisation des instances de composants

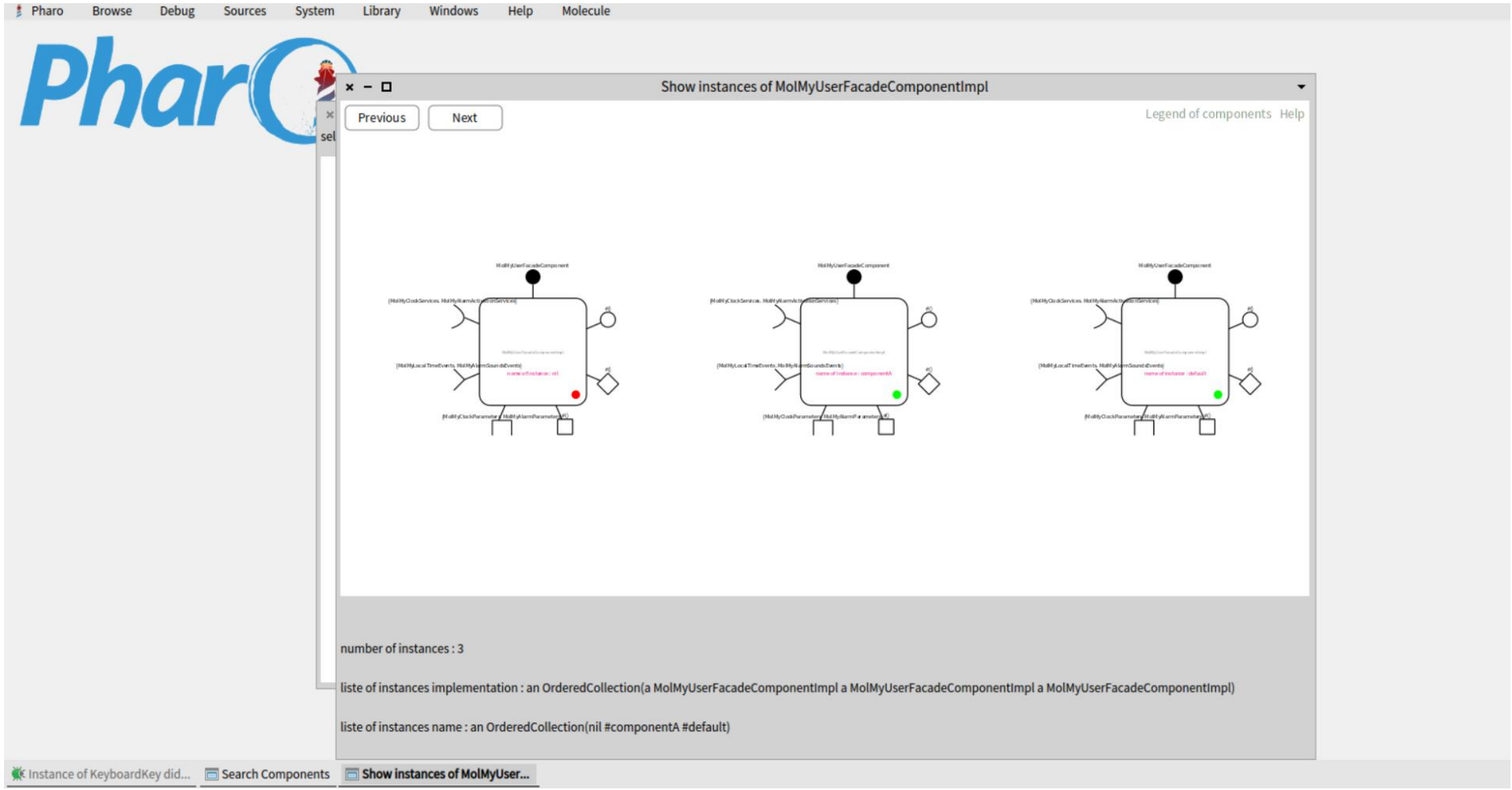

Figure 26 : Capture d'écran de la fenêtre des vues des instances de composants sur Pharo

Annexe XIII : Fenêtre correspondant à la visualisation de l'éditeur graphique

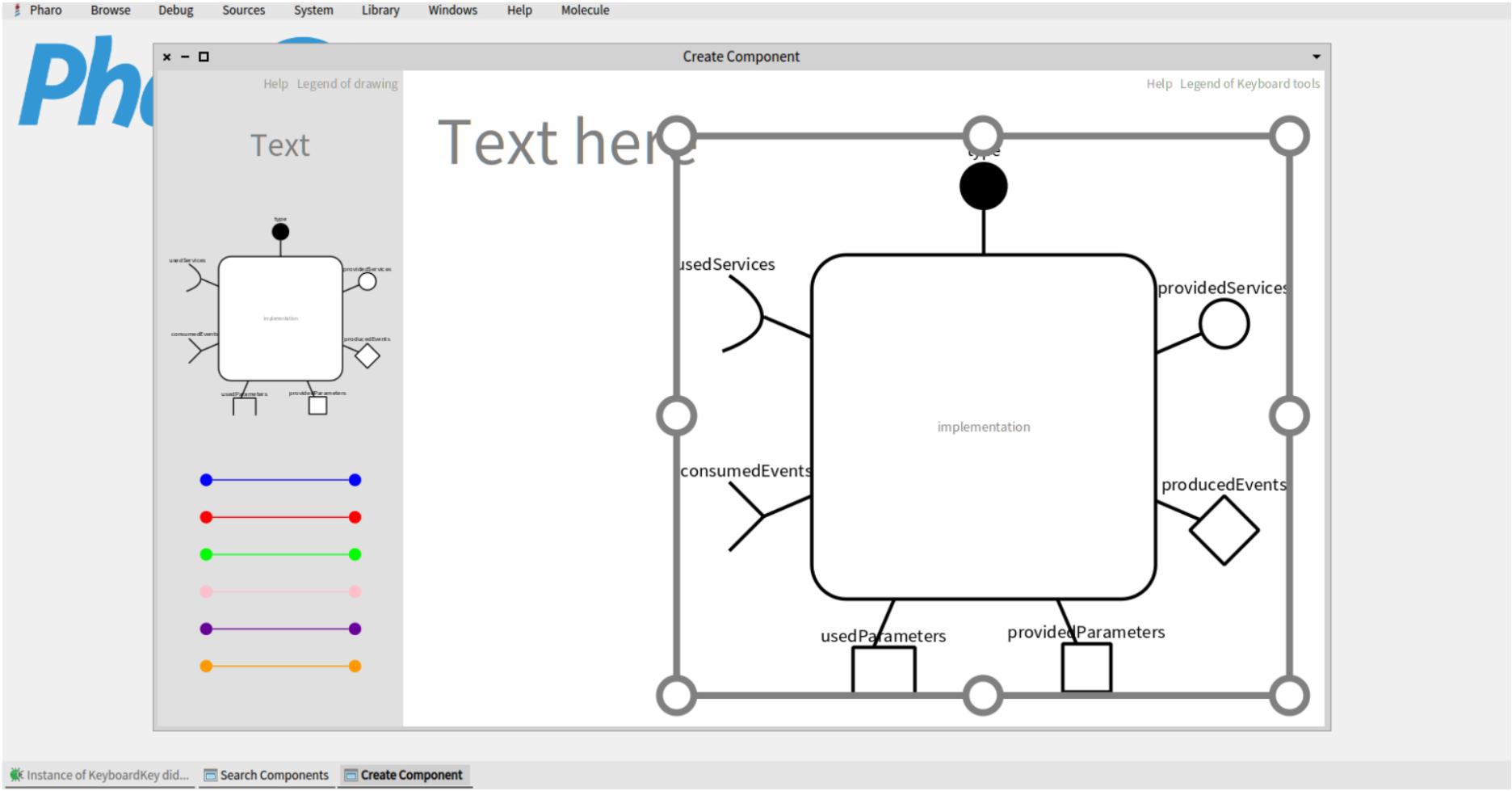

Figure 27 : Capture d'écran de la fenêtre correspondant à l'éditeur graphique sur Pharo

Tables des matières