Université du Québec
École de technologie supérieure

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS GTI792 PROJET DE FIN D'ÉTUDES EN GÉNIE DES TI]

DÉVELOPPEMENT D'UNE INTERFACE DE TÉLÉMÉTRIE ET DE CONTRÔLE POUR PLATEFORME AUTONOME

PIER-LUC CARON ST-PIERRE
CARP15098707

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur

ALAIN APRIL

MONTRÉAL, 13 AVRIL 2011
HIVER 2011

REMERCIEMENTS

Ce projet a pris forme grâce à l'implication des nombreux étudiants dans des projets qui permet de bâtir la réputation de l'ÉTS à travers le monde. Ce projet est bâti dans le cadre des activités du club S.O.N.I.A.

Simon Bolduc, le meneur de ce projet est membre du club étudiant S.O.N.I.A.. Des remerciements particuliers à Simon Bolduc pour avoir offert ce projet. Sa contribution a été hautement sollicitée durant toutes les étapes de réalisation du projet. Il a fourni une aide précieuse à tous les niveaux durant la réalisation de ce projet.

Par ailleurs, il est important de mentionner la confiance accordée par Martin Morissette, Jonathan Ducharme et Kevin Larose en tant que capitaine du club S.O.N.I.A. durant les années passées.

DÉVELOPPEMENT D'UNE INTERFACE DE TÉLÉMÉTRIE ET DE CONTRÔLE POUR PLATEFORME AUTONOME

PIER-LUC CARON ST-PIERRE
CARP15098707

RÉSUMÉ

L'objectif du projet est de développer une solution de communication en temps réel avec des systèmes autonomes. Cette solution sera compatible entre plusieurs plateformes robotiques existantes. Il devra être possible d'obtenir des données et d'envoyer des commandes à ces systèmes en temps réel. L'interface doit permettre à l'opérateur de bénéficier d'une compréhension plus rapide de la situation actuelle du robot.

Cette solution doit disposer de modules de personnalisation d'interface graphique. Il sera également possible de développer de nouveaux composants graphiques rapidement à l'aide d'une boîte à outils de développement. Des mécanismes sont développés afin de faciliter les activités de développement.

Une méthodologie basée sur OpenUP a été utilisée. Également, quelques éléments de conception architecturale ont été ajoutés en cours de projet.

TABLE DES MATIÈRES

Page

INTRODUCTION 1

CHAPITRE 1 CONTEXTE 2
1.1 Le club S.O.N.I.A. 2
1.2 Compétition de sous-marin autonome 3
1.3 Initiative Octets 4

CHAPITRE 2 DESCRIPTION DU PROJET 6
2.1 Problématique 6
2.2 Objectifs 7
2.2.1 Communication bidirectionnelle entre l'opérateur et le robot 7
2.2.2 Compatibilité assurée entre plusieurs systèmes autonomes 7
2.2.3 Développement accéléré des composants graphiques 8
2.2.4 Configuration flexible des composants graphiques 8
2.2.5 Personnalisation simple de l'interface de téléopération 8
2.2.6 Développement de prototypes de composants graphique. 9
2.3 Hypothèses et dépendances 9

CHAPITRE 3 MÉTHODOLOGIE 10
3.1 Objectifs de la méthodologie 10
3.1.1 Solution utilisable tôt dans le cycle de développement 10
3.1.2 Réduire la période d'intégration au système 10
3.1.3 Réduction des régressions 10
3.1.4 Rétroaction rapide 11
3.2 Développement continu 11
3.3 Intégration continue améliorée 12
3.4 Suivi des tâches 15
3.5 Approche générale 15

CHAPITRE 4 CHOIX TECHNOLOGIQUES, ARCHITECTURE ET CONCEPTION 18
4.1 Choix technologiques 18
4.1.1 Architecture JAUS 18
4.1.2 Plateforme Java 18
4.1.3 Boîte à outils Swing 18
4.1.4 Gestion de dépendance Maven 19
4.1.5 Spécialisation des modules 19
4.1.6 Internationalisation 19
4.1.7 Sérialisation avec YAML 19
4.2 Architecture 20
4.2.1 Architecture générique 20
4.2.2 Architecture spécialisée à S.O.N.I.A. 21

4.2.3 Déploiement pour activités de développement 22
4.2.4 Déploiement pour système autonome 23
4.3 Conception 24
4.3.1 Sous-système de configuration 24
4.3.2 Sous-système d'internationalisation 25
4.3.3 Injection de dépendances 26
4.3.4 Sauvegarde et chargement des perspectives 27
4.3.5 Isolation des composants graphiques 28

CHAPITRE 5 RÉSULTATS 29
5.1 Travaux réalisés 29
5.1.1 Document de vision (DE6) 29
5.1.2 Modèle du domaine (DE8) 29
5.1.3 Glossaire (DE9) 29
5.1.4 Diagramme des cas d'utilisation (DE12) 30
5.1.5 Cas d'utilisation (DE13) 30
5.1.6 Maquettes fil de fer (DE14) 30
5.1.7 Cahier d'architecture (DE15) 30
5.1.8 Scénarios de qualités logiciels (DE17) 31
5.2 Résultats 31
5.2.1 Communication bidirectionnelle entre l'opérateur et le robot 31
5.2.2 Compatibilité assurée entre plusieurs systèmes autonomes 32
5.2.3 Développement accéléré des composants graphiques 33
5.2.4 Configuration flexible des composants graphiques 33
5.2.5 Personnalisation simple de l'interface de téléopération 34
5.2.6 Développement de prototypes de composants graphique. 36

CONCLUSION 37

RECOMMANDATIONS 38

BIBLIOGRAPHIE 39

ANNEXE I DOCUMENT DE VISION 41

ANNEXE II MODÈLE DU DOMAINE 48

ANNEXE III GLOSSAIRE 50

ANNEXE IV DIAGRAMME DE CAS D'UTILISATION 52

ANNEXE V CAS D'UTILISATIONS 53

ANNEXE VI MAQUETTES FIL DE FER 58

ANNEXE VII SCÉNARIOS DE QUALITÉ.....60

ANNEXE VIII CAHIER D'ARCHITECTURE.....62

ANNEXE IX DOCUMENT DE CONCEPTION 77

LISTE DES TABLEAUX

Page

Tableau 3-1 Système et technologie utilisée.....14

LISTE DES FIGURES

Page

Figure 1-1 — Prototype 2009-2010 et 2011 2

Figure 1-2 — Parcours de la compétition 2010 4

Figure 3-1 — Développement continu 12

Figure 3-2 — Intégration continue amélioré 14

Figure 4-1 — Diagramme en couche de l'architecture générique 20

Figure 4-2 — Diagramme en couche de l'architecture spécialisée pour le club S.O.N.I.A. 21

Figure 4-3 — Vue de déploiement pour activités de développement 22

Figure 4-4— Vue de déploiement pour système autonome 23

Figure 4-5 — Diagramme de classe du sous-système de configuration 24

Figure 4-6— Diagramme de classe du sous-système d'internationalisation 25

Figure 4-7— Diagrammes de classe du sous-système d'injection de dépendance 26

Figure 4-8 — Diagramme de classe du sous-système d'internationalisation 27

Figure 4-9— Diagramme de classe du sous-système de composants graphiques 28

Figure 5-1— Communication bidirectionnelle entre l'opérateur et le robot 32

Figure 5-2 — Liste des thèmes disponibles 35

Figure 5-3 — Sauvegarde et chargement de la disposition de l'interface graphique en anglais 35

Figure 5-4 — Sauvegarde et chargement de la disposition de l'interface graphique en français 36

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AUV6	Système de contrôle autonome du club S.O.N.I.A.
AUVSI	Association for Unmanned Vehicle Systems International
CAN	Controller Area Network
ÉTS	École de technologie supérieure
JAUS	Joint Architecture for Unmanned Systems
S.O.N.I.A.	Système d'opération nautique intelligent et autonome
YAML	YAML Ain't Markup Language

INTRODUCTION

De nos jours, les robots effectuent des tâches impossibles pour les humains. Que ce soit l'inspection de barrage ou la capture d'image d'une centrale nucléaire après un tsunami, les robots sont appelés à effectuer toute sorte de tâches périlleuses.

Lorsque l'on développe un système conscient de son environnement, il est essentiel pour l'opérateur du système de connaître la situation perçue par ce système. Afin d'accomplir cet objectif, le club S.O.N.I.A. a développé au cours des années une solution de téléopération afin de recevoir les données captées par le sous-marin autonome.

Un système de téléopération doit pouvoir recevoir et envoyer des données. En plus de communiquer de façon bidirectionnelle avec le robot, ce système doit afficher les données de façon conviviale afin que l'utilisateur puisse facilement comprendre ces données. Ce problème est également présent lors d'envoi de directives au robot. L'opérateur doit envoyer la bonne commande dans des délais raisonnables afin d'effectuer diverses opérations.

Ce projet a pour objectif de développer un système de téléopération efficace qui sera utilisable sur plusieurs plateformes autonomes. En premier lieu, une explication du contexte de réalisation du problème sera exposée. Par la suite, une description du projet sera présentée. Ensuite, la méthodologie utilisée afin de réaliser le projet sera expliquée. Puis, les choix technologiques, l'architecture et la conception seront survolés. Finalement, le document terminera avec la présentation des résultats.

CHAPITRE 1

CONTEXTE

1.1 Le club S.O.N.I.A.

Le club S.O.N.I.A. est un regroupement étudiant de l'ÉTS à vocation technique. L'objectif principal du club est de développer un sous-marin autonome afin de participer à une compétition étudiante. Le club a également comme objectif d'offrir une expérience de réalisation de projet aux étudiants afin d'enrichir la formation de ses membres.

Le club S.O.N.I.A. est une équipe multidisciplinaire qui développe des prototypes afin de réaliser les objectifs de la compétition. Il faut donc mentionner que le robot construit est composé de prototypes dans tous les domaines du génie. Essentiellement, l'équipe mécanique doit s'assurer de l'étanchéité du robot. Les membres de l'équipe électronique s'occupent de l'alimentation et de la communication entre les composants électroniques. Finalement, l'équipe logicielle doit développer le système de décision autonome ainsi que tout autre outil pouvant assister l'accomplissement des objectifs.

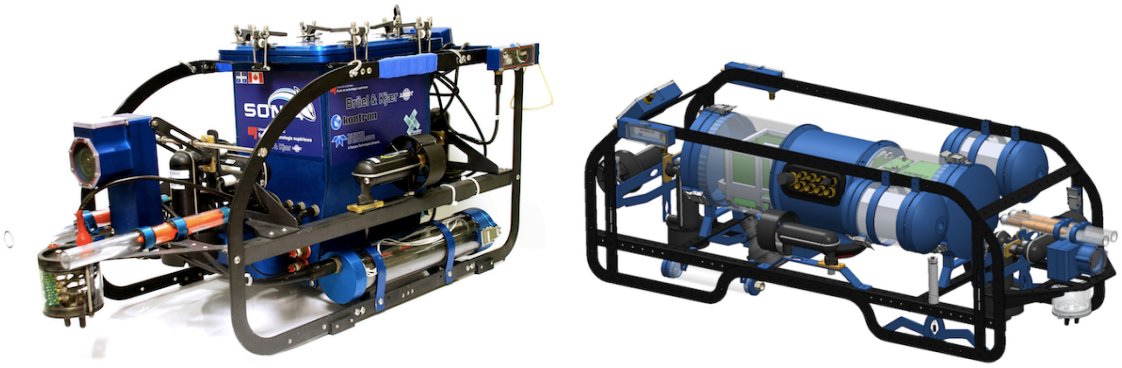


Figure 1-1 — Prototype 2009-2010 et 2011

1.2 Compétition de sous-marin autonome

La compétition dont participe le club S.O.N.I.A. est organisée annuellement par «The AUVSI Foundation ». Cette compétition représente des tâches réelles que les robots sous-marins autonomes déployés ont à effectuer. Cette compétition se déroule annuellement vers la fin juillet à San Diego. Cette ville se situe dans le sud de la Californie. La compétition a une portée internationale. Cette année l'évènement pourra compter sur la participation d'équipe de l'Asie et de l'Europe. La compétition comporte également un nombre élevé d'équipe de chaque coin des États-Unis et du Canada.

Lors de la compétition, le sous-marin doit être en mesure de reconnaître son environnement et de mener avec succès des actions précises. Il doit être également capable de naviguer au travers du parcours et de prendre des décisions selon ce qu'il capte de son environnement.

La compétition consiste à une mission que le sous-marin a à réaliser. Chaque mission est composée d'obstacles. Chacun des obstacles si effectués avec succès ou partiellement donne un certain nombre de points. L'objectif est donc de faire plus de points que les autres équipes. Voici une illustration représentant le parcours à effectuer lors de la compétition 2010 :

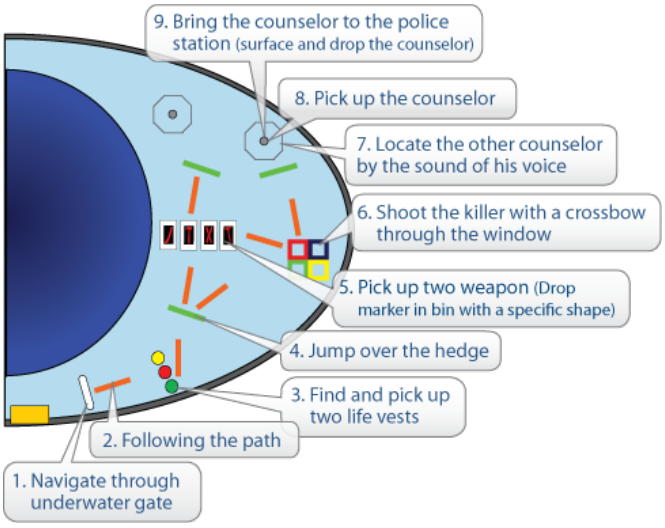


Figure 1-2 — Parcours de la compétition 2010

Durant les dernières années, le club S.O.N.I.A. a réussi à se démarquer des autres universités par sa compétitivité, son professionnalisme et l'esprit d'entraide qu'il apporte aux autres équipes. L'équipe a également réussi à décrocher un podium à plusieurs reprises.

1.3 Initiative Octets

L'initiative Octets est une expérience ayant débuté lors de l'automne 2010. L'objectif de l'initiative est de permettre une plus grande coopération entre les différents regroupements scientifiques de robotique de l'ÉTS. Cette organisation regroupe les clubs Capra, Dronolab, S.O.N.I.A. et Walking Machine.

Le modèle de coopération est basé sur les modèles des projets à source libre. Tous les membres des clubs sont appelés à utiliser les projets à contribuer aux projets en apportant des améliorations.

L'idée derrière cette initiative était de partager les efforts de développement et de produire des solutions logicielles de meilleure qualité. De plus, le partage des connaissances permet de

réduire la duplication des efforts de développement. Beaucoup de défis techniques sont communs à chacun des clubs.

Les projets touchés par l'initiative sont les projets concernant les tâches de reconnaissances visuelles, l'utilisation du protocole de communication CAN, ainsi que les différents projets reposants sur l'architecture JAUS.

CHAPITRE 2

DESCRIPTION DU PROJET

2.1 Problématique

Lorsque l'on développe un système conscient de son environnement, il est essentiel pour l'opérateur du système de connaître la situation perçue par ce système. Afin d'accomplir cet objectif, le club S.O.N.I.A. a développé au cours des années une solution de téléopération afin de recevoir les données captées par le sous-marin autonome. Ce système permet également à l'opérateur d'envoyer des commandes au robot afin que celui interagissent avec son environnement.

Ainsi, un système de téléopération doit pouvoir recevoir et envoyer des données. En plus de communiquer de façon bidirectionnelle avec le robot, ce système doit afficher les données de façon conviviale afin que l'utilisateur puisse facilement comprendre ces données. Ce problème est également présent lors d'envoi de directives au robot. L'opérateur doit envoyer la bonne commande dans des délais raisonnables afin d'effectuer diverses opérations.

Récemment, l'équipe S.O.N.I.A. a décidé de partager plusieurs de ces projets avec les clubs scientifiques Capra, Dronolab et Walking Machine dans le cadre de l'initiative Octets. Chacune des organisations peut bénéficier d'une solution de téléopération. Cependant, la solution de téléopération actuelle développée par S.O.N.I.A. ne correspond pas aux besoins des autres clubs.

Parallèlement, chacun des regroupements étudiants a des besoins communs sur les fonctionnalités d'une interface de téléopération. Paradoxalement, chacun de ces groupes a des besoins spécifiques à leur robot et aux défis à réaliser. Il est donc nécessaire pour chaque

organisation de pouvoir étendre les fonctionnalités de la solution avec leurs propres composants.

Il est possible de faire un parallèle avec les modèles d'affaire «open core» des solutions logiciel libre. L'interface de télémétrie est un projet commun n'appartenant à aucun club spécifique. Cependant, chacun des regroupements ajoute leur propre fonctionnalité afin de répondre à leur besoin spécifique.

Les clubs scientifiques nécessitent donc une solution de téléopération fonctionnelle, modulaire et extensible répondant à leurs besoins.

2.2 Objectifs

En cours du projet, les objectifs ont évolué de plusieurs façons. La cadre du projet a été resituée afin de répondre aux critères de qualités logiciels espérer d'une solution de ce calibre. Afin de visualiser ses objectifs, voici la liste des objectifs établis ainsi qu'une courte description de chacun des objectifs.

2.2.1 Communication bidirectionnelle entre l'opérateur et le robot

L'objectif principal d'une interface de téléopération est de permettre l'opération à distance d'un robot. Afin de réaliser cette tâche, l'opérateur à besoin des données que le robot perçoit. Il doit également avoir la capacité d'envoyer des commandes au robot. En disposant d'une communication bidirectionnelle, l'opérateur pourra réaliser ces tâches.

2.2.2 Compatibilité assurée entre plusieurs systèmes autonomes

La solution est bâtie afin que chacun des clubs robotiques puisse avoir accès à une interface de téléopération avec leur robot. Afin de réaliser ce but, il est essentiel que la solution soit

utilisable afin de communiquer avec plusieurs systèmes autonomes. Le système pourra servir à communiquer avec tout système autonome compatible avec l'architecture JAUS.

2.2.3 Développement accéléré des composants graphiques

Une subtilité à considérer par l'aspect de la comptabilité entre systèmes autonomes est que chaque robot a des objectifs, des capteurs et des actuateurs différents. Le système devra donc offrir des mécanismes de développement de composant graphique. Ainsi, toute section de code réutilisable entre les clubs pourra l'être. Également, certains composants graphiques sont utilisés à tous les clubs. En fournissant des mécanismes intégrés, le développement de l'interface graphique sera accéléré et agréable à mener par les développeurs.

2.2.4 Configuration flexible des composants graphiques

L'objectif est de bâtir des composants graphiques flexibles facilement configurables. Cet objectif vise à augmenter le degré de réutilisation. Un sous-aspect de cet objectif est de rendre l'interface graphique multilingue. Bref, cet objectif rendra l'application plus agréable à utiliser et à développer.

2.2.5 Personnalisation simple de l'interface de téléopération

L'interface de téléopération doit être personnalisable par les utilisateurs. L'interface devra être en mesure d'accepter un certain nombre de personnalisations tel que le thème de couleurs et le positionnement des fenêtres. Ces personnalisations doivent être sauvegardable et récupérables durant l'exécution de l'application. Ainsi, l'application permettra aux utilisateurs de faire une configuration et de la réutiliser à chaque utilisation plutôt que de reconfigurer à chaque fois son interface.

2.2.6 Développement de prototypes de composants graphique.

Le projet consiste à développer la base d'un système auquel il sera ajouté des composants. Afin de savoir si le cadriciel répond au besoin, il est essentiel de tester les mécanismes fournis. Ainsi, une certaine partie du projet est réservé au développement de prototypes de composants graphiques. Cette partie servira à évaluer les mécanismes développés au cours du projet.

2.3 Hypothèses et dépendances

Tel que décrit dans le cahier d'architecture, le projet repose sur l'hypothèse suivante :

- Les utilisateurs ont de bonnes intentions.

Cette hypothèse suggère que les utilisateurs de l'application ont de bonnes intentions. Ainsi, il est possible de négliger les comportements malveillants lors de la conception de l'application. Cependant, il faut veiller à ce qu'un utilisateur ou un développeur ne commette par d'erreurs par inattention ou par manque de connaissance.

De plus, le système possède la dépendance suivante :

- Une bibliothèque applicative permettant d'échanger des messages entre systèmes autonomes en utilisant l'architecture JAUS existe et est sans problème.

Le système repose entièrement sur cette bibliothèque applicative pour les mécanismes de communication avec le système autonome. Ainsi, cette dernière possède des anomalies, le système risque également de comporter des anomalies. Il est implicite que la bibliothèque applicative est complète afin d'assurer l'intégrité des fonctionnalités.

CHAPITRE 3

MÉTHODOLOGIE

3.1 Objectifs de la méthodologie

La méthodologie de développement a été utilisée afin de remplir les objectifs suivants.

3.1.1 Solution utilisable tôt dans le cycle de développement

La solution de téléopération permet à l'opérateur de communiquer et d'envoyer des commandes au prototype. Ces activités de communication sont critiques lors des périodes de tests. Si la solution n'est pas utilisable, beaucoup de temps sont perdus à ne pas pouvoir tester les autres sous-systèmes du robot.

3.1.2 Réduire la période d'intégration au système

Cette année, la majorité des sous-systèmes de S.O.N.I.A. subissent un réusinage d'envergure. Tout au long de l'année, de nouveaux sous-systèmes seront développés. L'approche utilisée vise à connaître les coûts d'intégration au cours du développemement de la solution.

3.1.3 Réduction des régressions

L'approche vise également à réduire les risques de régressions. Ainsi, la solution doit toujours être en prêt à l'utilisation. En règle générale, les clubs étudiants n'ont pas de notion de mise en production. Les systèmes utilisés sont toujours les dernières versions développées. Il est donc essentiel de réduire les risques de régression.

3.1.4 Rétroaction rapide

Un dernier objectif de l'approche utilisée était de pouvoir avoir une rétroaction rapide de la part des développeurs et des utilisateurs. En obtenant une rétroaction rapide, il est possible de corriger au fur et à mesure le problème.

Ces objectifs ont contribué à développer l'approche utilisée

3.2 Développement continu

Plusieurs projets sont menés en parallèle afin que chacune des équipes puisse répondre à leurs objectifs. Ainsi, durant le développement des fonctionnalités couvertes par ce projet, le système continue à évoluer au fil du temps. Il est donc important d'appliquer une discipline au cours du développement afin de s'assurer de ne rien rendre inutilisable. Également, les gens utilisent la suite logicielle afin d'ajouter leurs propres fonctionnalités. Certaines de ses fonctionnalités ne sont pas partagées par d'autres groupes. Il est important d'assurer de communiquer les changements et de migrer doucement lors de changements venant briser la rétrocompatibilité. Il faut donc assurer une compatibilité ascendante et descendante temporellement limitée.

Voici une illustration afin de bien comprendre le cycle de développement.

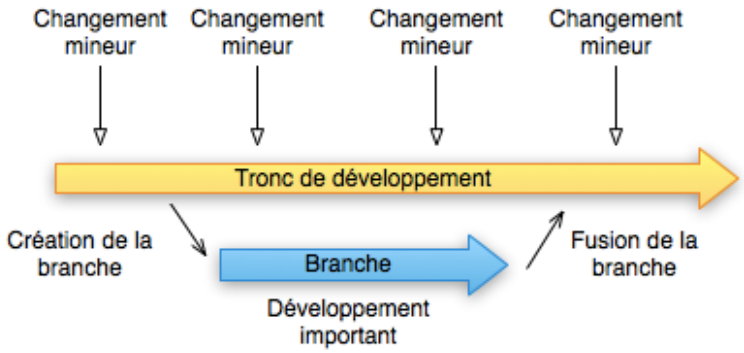


Figure 3-1 — Développement continu

Comme l'illustre l'image, le développement se déroule sur le tronc de développement. Plusieurs changements mineurs peuvent survenir au fil du temps sur le tronc de développement. Cependant, lorsqu'un changement majeur est envisagé, les développeurs créent une branche de développement. De cette façon, les travaux sur le tronc de développement peuvent se poursuivre en même temps que le développement majeur sur une branche. Lorsque l'ajout est mature, la branche est réintégrée au tronc. Elle est alors fusionnée avec le tronc. Cette approche d'équipe est doublement importante. De façon générale, les utilisateurs des applications exécutent le code qui se trouve sur le tronc de développement. Il n'y a pas de système de de mise en production, car le système évolue trop rapidement. Les développeurs sont donc tenus de porter une attention particulière aux changements apportés au tronc.

3.3 Intégration continue améliorée

D'un autre côté, la pratique d'intégration continue a été instaurée au sein du groupe de développeur. Cette pratique consiste à la mise en place en plusieurs mécanismes permettant une intégration continue améliorée.

Premièrement, un système de contrôle de version centralisé est disponible afin de contenir le code ainsi que les modifications apportées. Cette pratique a comme principal avantage de permettre au développeur d'échanger facilement leur code. Également, un système de contrôle de version permet aux développeurs de consulter les versions précédentes afin de récupérer des fragments ayant été supprimés. Cette mesure facile la récupération des erreurs lorsqu'une régression est produite.

En second lieu, lorsqu'un développeur effectue une publication de modifications, un courriel est envoyé à une liste de distribution. Cette liste de distribution regroupe les développeurs de

l'initiative Octets. Le courriel envoyé contient des informations pertinentes au sujet de la publication de la modification telle que le code ajouté, modifié et supprimé ainsi que le commentaire attaché à la publication de modification. Cette pratique permet aux développeurs d'être informés des ajouts de fonctionnalités et des modifications au code. Également, elle permet aussi aussi aux développeurs d'effectuer une revue de code au fur et à mesure au développement. Finalement, cette pratique incite les développeurs à discuter des modifications.

Également, un second mécanisme s'enclenche lorsqu'un contributeur effectue une publication de modification. Un système d'intégration continue reçoit une notification de la part du système de contrôle de version. À la réception de cette notification, le système d'intégration continue enclenche la construction du logiciel à partir de la dernière version disponible dans le système de contrôle de version. Si la construction est un succès, le système va alors lancer les tests automatisés. Si l'une de ces deux étapes échoue, un courriel est envoyé aux développeurs afin de les informer de la situation. Idéalement, les développeurs corrigent la situation dans l'immédiat.

De plus, lorsque les tests ont été exécutés avec succès, le système d'intégration rend disponible le résultat de la compilation. Ces résultats sont nommés artefacts. Le système publie les artefacts dans un système de gestion d'artefacts. Une fois publiés, ces artefacts sont disponibles pour utilisation.

Lors de la compilation, du développement et du déploiement des projets, un système de gestion de production logiciel utilise les dépendances archivées. Ce système va contacter le système de gestion d'artefacts afin d'obtenir les artefacts nécessaires à l'exécution de l'application. Ainsi, la gestion des dépendances est simplifiée, car elle est encapsulée par un système spécialisé dans la gestion des dépendances.

Cette image illustrant le cycle de vie complet accompagné les technologies actuelles utilisés :

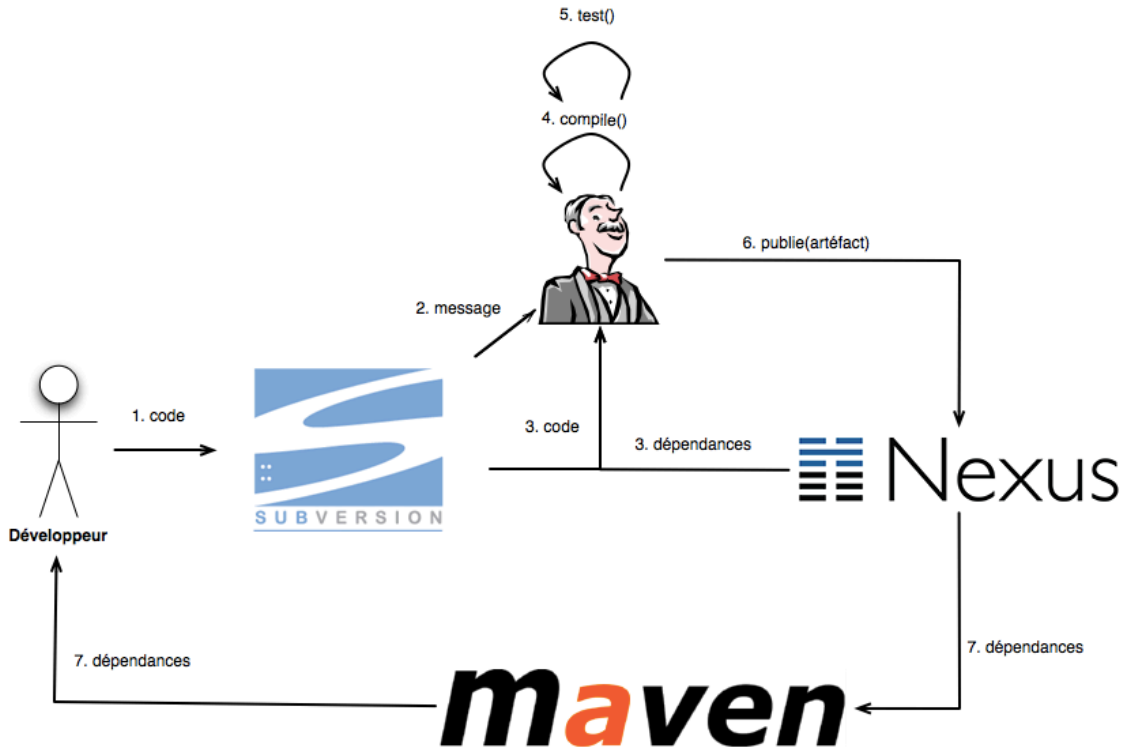


Figure 3-2 — Intégration continue amélioré

Système	Technologie
Système de contrôle de version	Subversion
Système d'intégration continue	Jenkins
Système de gestion d'artefacts	Nexus
Système de gestion de production logiciel	Maven

Tableau 3-1 Système et technologie utilisée

3.4 Suivi des tâches

Dans le cadre du développement du système, plusieurs tâches ont dû être menées à terme. Les trois catégories de tâches typiques sont la correction d'anomalies, l'amélioration de fonctionnalité existante ou l'ajout de nouvelles fonctionnalités.

Dans chacun des cas, un billet est créé dans un système de gestion de billets afin d'assurer le suivi du développement à effectuer. Ainsi, une entité permettant de faire le suivi du développement est créée. Cette entité est utilisée afin de signifier toute incohérence de la demande et pour indiquer l'état de complétion de la tâche.

De plus, il est possible d'associer des observateurs aux billets. Ainsi lorsque des modifications sont apportées aux billets, les observateurs sont immédiatement informés par courriel. Cette fonctionnalité permet d'assurer le suivi automatique pour le requérant et le développeur de la tâche.

Il est important de noter que cette approche est seulement utilisée afin de faire un suivi des activités de développement. Ce système ne cherche pas à servir comme outils de contrôle des activités. La communication de vive voix est toujours favorisée.

3.5 Approche générale

L'approche générale afin de développer a été décomposée en plusieurs étapes.

D'abord, un document de vision a été réalisé. Ce document a permis de définir la portée du projet et de s'entendre sur les lignes directrices du projet.

Ensuite, un glossaire a été construit en même temps qu'un modèle du domaine. Ces artefacts ont permis de bien comprendre l'espace du problème. Une grande partie des termes avancés était reliée à l'architecture JAUS.

Quelques cas d'utilisation et scénarios de qualité ont été rédigés afin d'éclaircir certains points aux niveaux des résultats escomptés par le projet.

Par la suite, une ébauche d'architecture a été développée. Cette ébauche a permis de tirer les grandes lignes du système. À cette étape, l'activité la plus risquée en terme d'implémentation a été lancée. Les télémétries ont réussi à être spécialisées afin que chacun des clubs ait sa propre télémétrie bâtie à partir de la télémétrie commune. À ce moment les composants spécifiques pour chacun des clubs ont déplacé du projet commun vers les télémétries spécialisé.

Par la suite, une conception un peu plus élaborée a été développée. Cette conception a été mise à l'épreuve en l'implémentant. Cette phase d'implémentation a permis de tester les décisions et de s'assurer que les choix étaient les meilleurs.

Les sous-systèmes de la téléopération ont souvent été revisités afin de s'assurer que les approches utilisées étaient les meilleures. Par exemple, la gestion des groupes de composants a été revisitée trois fois. À chaque fois la conception a été améliorée. Cette approche comporte plusieurs avantages. Premièrement, elle permet d'obtenir un produit de plus en plus solide. En second lieu, elle permet au développeur de construire par dessus ce qui est prouvé fonctionnel. Idéalement, les fonctionnalités de base ont été testées entre chaque itération. Également, les utilisateurs peuvent jouir rapidement des nouvelles fonctionnalités qui seront bonifiées au fil du temps.

Finalement, l'architecture a été formalisée en développant un cahier d'architecture décrivant les différents choix. Ce document a été rédigé dans l'espoir de pouvoir justifier les choix pris.

Ces choix ne sont pas définitifs et seront revisités dans le futur. Ils ont seulement été documentés afin d'expliquer sur quelles hypothèses ses choix ont été faits. Il regroupe également le différent compromis pris au sujet de la conception des composants.

CHAPITRE 4

CHOIX TECHNOLOGIQUES, ARCHITECTURE ET CONCEPTION

4.1 Choix technologiques

Cette section présente les différents choix technologiques. Le cahier d'architecture contient plus de détails sur les raisons derrière ces choix.

4.1.1 Architecture JAUS

L'architecture JAUS a été retenue afin d'assurer la communication entre le système autonome et le système de téléopération. Les principales raisons sont que l'implémentation de JAUS est une épreuve de compétition. Également, la norme est reconnue dans le domaine des systèmes autonomes.

4.1.2 Plateforme Java

La plateforme Java a été retenue afin de développer le système. Les principales raisons sont que la plateforme est utilisée dans le cadre des cours à l'ÉTS. Également, la plupart des systèmes du club sont développés en Java. Finalement, la plateforme Java contient plusieurs mécanismes automatiques facilitant le développement.

4.1.3 Boîte à outils Swing

La boîte à outils Swing a été utilisée afin de réaliser le système. Les principales raisons sont que Swing est intégré à java. Également, les membres du club S.O.N.I.A. possèdent déjà une expérience de construction d'application avec la boîte à outils Swing.

4.1.4 Gestion de dépendance Maven

Le système de gestion de dépendance Maven a été retenu afin d'assurer la gestion des librairies. Les principales raisons est que Maven est reconnu dans l'industrie et qu'il permet de simplifier la gestion des dépendances lors des activités de développement.

4.1.5 Spécialisation des modules

L'approche spécialisation de module a été utilisée afin de permettre une réutilisation maximale des composants déjà développés. Également, l'approche permet de séparer les responsabilités organisationnelles.

4.1.6 Internationalisation

Il a été décidé d'offrir le support à l'internationalisation. Cette décision a été prise afin d'accommoder les différentes situations où la télémétrie est présentée publiquement. Par exemple, plusieurs démonstrations sont effectuées à Montréal avec un public francophone. Cependant, une démonstration est demandée à la compétition à San Diego. Il faut donc une interface anglaise.

4.1.7 Sérialisation avec YAML

La sérialisation YAML a été utilisée pour deux principales raisons. Premièrement, elle permet de contrôler les informations qui sont sérialisées contrairement à l'API de sérialisation de Java. Également, un fichier en format YAML est clair et peut être compris et modifié facilement par un utilisateur

4.2 Architecture

Cette section présente un survol des artefacts d'architecture développés. Le cahier d'architecture contient une description détaillée de chacun des éléments.

4.2.1 Architecture générique

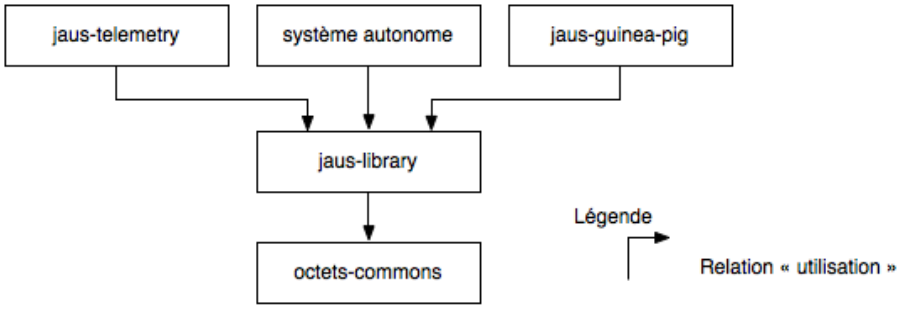


Figure 4-1 — Diagramme en couche de l'architecture générique

Cette architecture est une implémentation de base d'une infrastructure JAUS permettant la communication entre différents composants. Chaque composant pouvant communiquer JAUS a besoin de la jaus-library, car celle-ci contient les mécanismes nécessaires à l'utilisation de JAUS. La couche octets-commons permet d'offrir des services communs à chacune des applications.

4.2.2 Architecture spécialisée à S.O.N.I.A.

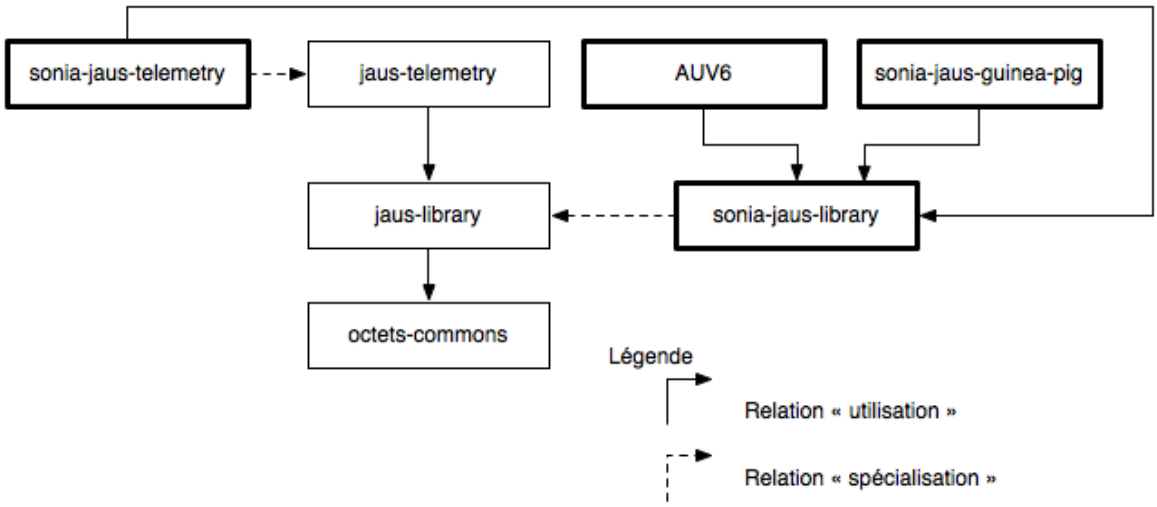


Figure 4-2 — Diagramme en couche de l'architecture spécialisée pour le club S.O.N.I.A.

Ce diagramme présente l'architecture spécialisée avec les systèmes S.O.N.I.A. Ce diagramme est semblable au précédent. Cependant, on lui a ajouté les entités spécifiques au clubs S.O.N.I.A. Le système sonia-jaus-library ajoute des messages spécifiques au système jaus-library. Ces messages spécifiques des informations telles que les données du capteur du sous-marin. Le système sonia-jaus-telemetry est la jaus-telemetry avec des composants graphiques de plus. Ces composants servent à présenter les données du sous-marin.

4.2.3 Déploiement pour activités de développement

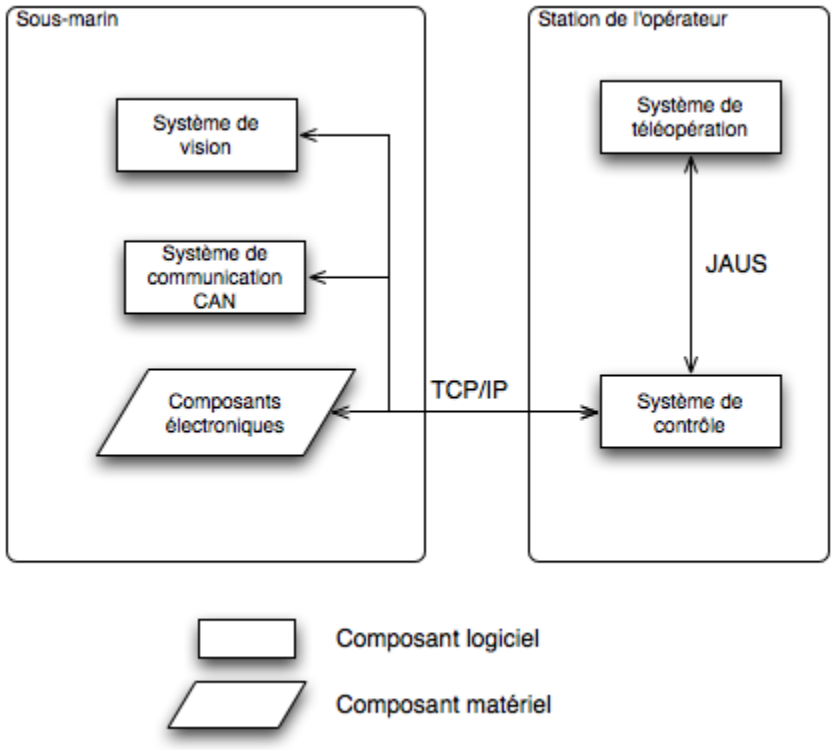


Figure 4-3 — Vue de déploiement pour activités de développement

Cette vue illustre le véhicule en mode de développement. Le système de contrôle est placé sur le système de l'opérateur afin de faciliter le développement. Cependant, cette configuration peut limiter les performances.

4.2.4 Déploiement pour système autonome

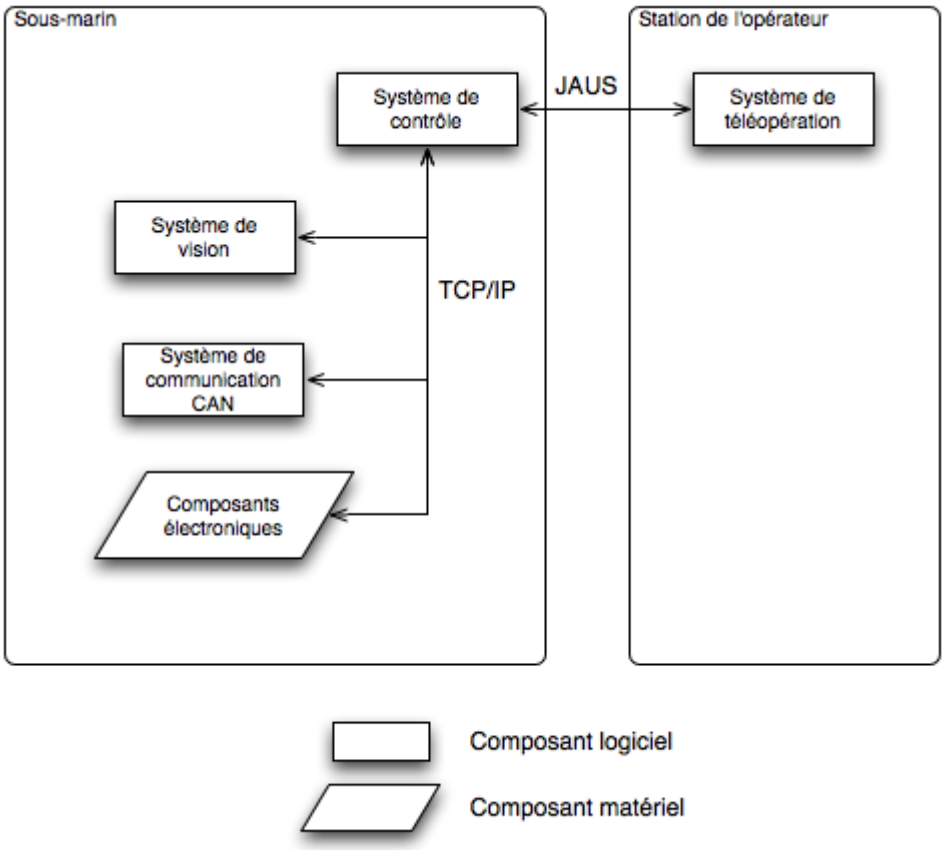


Figure 4-4— Vue de déploiement pour système autonome

Cette configuration de déploiement est la configuration afin d'avoir un système autonome. Il est possible de couper le lien entre la station de l'opération et le sous-marin afin d'obtenir un système complètement autonome. Cette approche a pour avantage d'isoler le contrôle de l'opérateur. Également, les problèmes de communication réseau n'empêcheront pas le système de se contrôler.

4.3 Conception

Cette section présente un survol des artefacts de conception développés. Le document de conception contient une description détaillée de chacun des éléments.

4.3.1 Sous-système de configuration

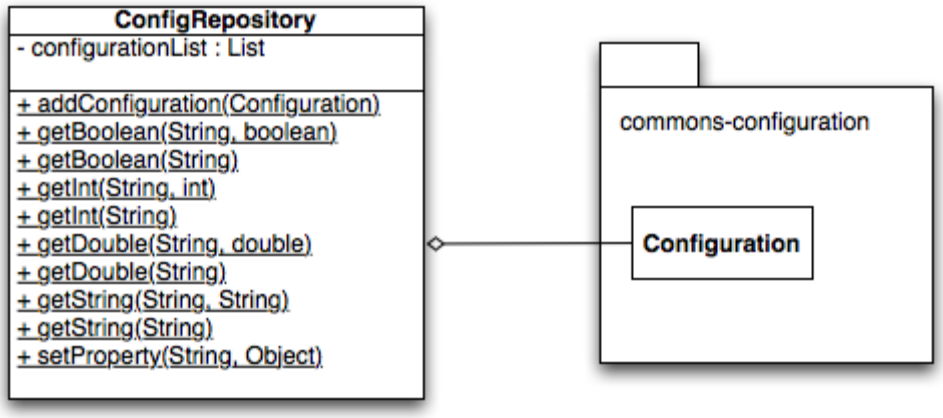


Figure 4-5 — Diagramme de classe du sous-système de configuration

Cette conception permet d'encapsuler l'utilisation de la librairie commons-configuration offert par la communauté Apache. Également, cette conception offre au développeur un point d'accès unique afin de manipuler et d'obtenir les valeurs de configuration durant le développement.

4.3.2 Sous-système d'internationalisation

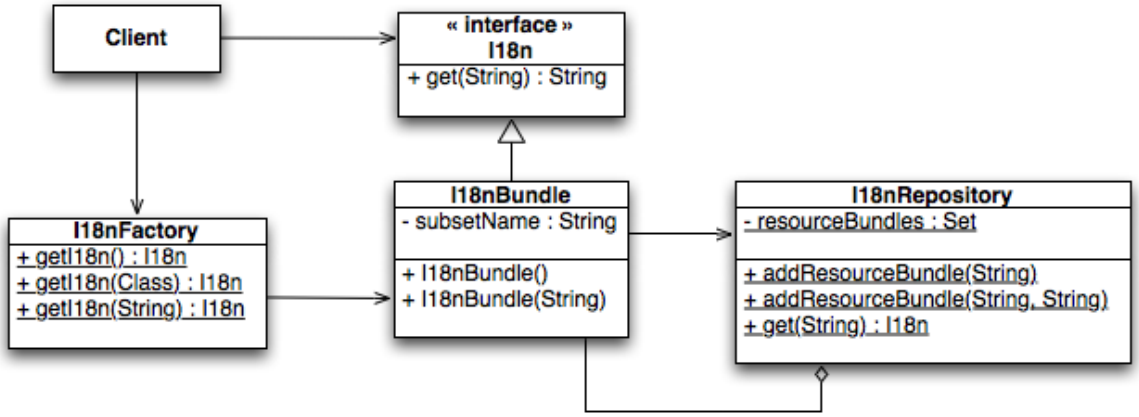


Figure 4-6— Diagramme de classe du sous-système d'internationalisation

Cette solution permet au développement d'utiliser le système d'internationalisation. Elle permet d'obtenir la valeur des clefs associées aux chaines de caractère de façon localisé. Également, il est possible de créer un sous-ensemble afin de rendre la manipulation plus conviviale lors du développement de l'interface graphique.

4.3.3 Injection de dépendances

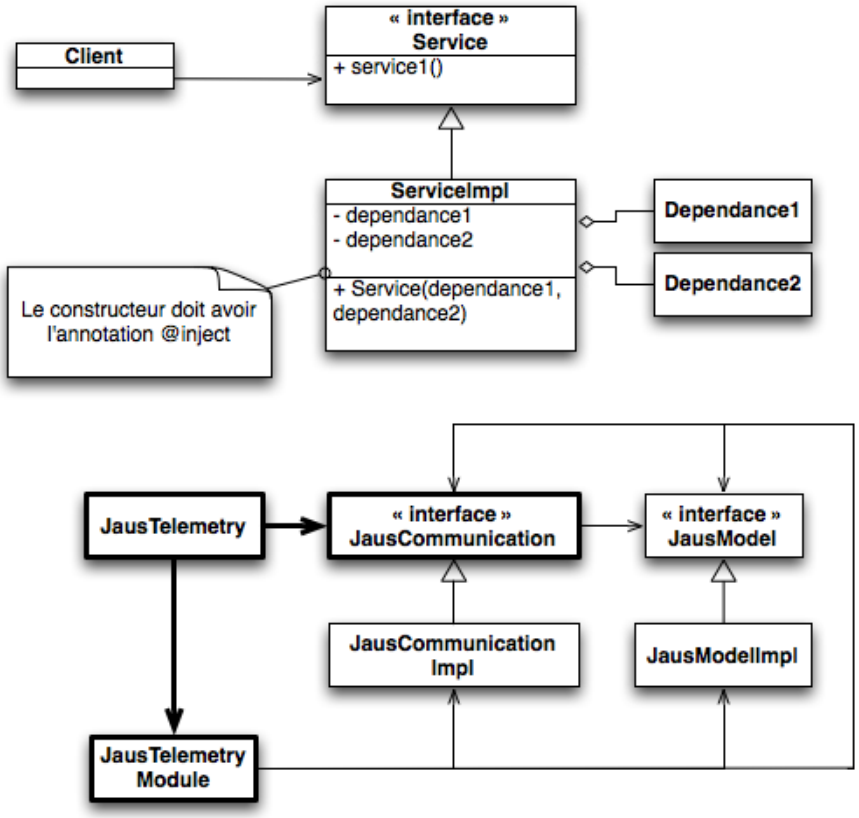


Figure 4-7— Diagrammes de classe du sous-système d'injection de dépendance

L'injection de dépendance permet de faire de l'inversion de contrôle lors de notre conception. Cette conception repose sur l'utilisation de la bibliothèque applicative Google Guice. Il est important de noter que la classe cliente est seulement couplée à l'interface du service et non de son implémentation. Également, la classe cliente ne connait pas les classes et interface auquel le service dépend. De cette façon, le système offre une modularité des composants permettant d'augmenter la testabilité. Également, l'approche vient réduire le couplage entre les classes.

4.3.4 Sauvegarde et chargement des perspectives

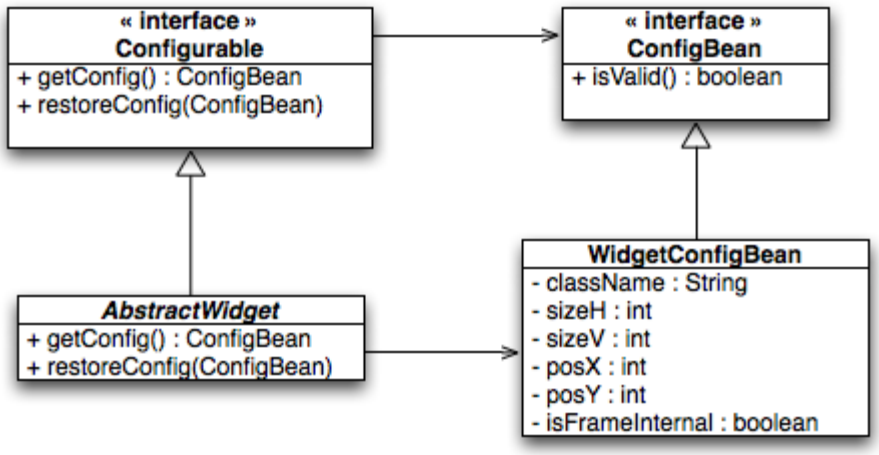


Figure 4-8 — Diagramme de classe du sous-système d'internationalisation

Cette conception est essentiellement une adaptation du patron de conception memento. La classe AbstractWidget crée des instances de WidgetConfigBean représentant sa configuration. Ces instances sont alors sérialisées par la classe cliente en format YAML. Également, il est possible pour AbstractWidget de restaurer une configuration. Les interfaces ont été ajoutées afin de guider le développement. En implémentant l'interface Configurable, la classe indique qu'elle fournit les services permettant de créer et de restaurer son état. La configuration offre un service de validation afin de s'assurer que les données sont valides avant de restaurer des informations invalides.

4.3.5 Isolation des composants graphiques

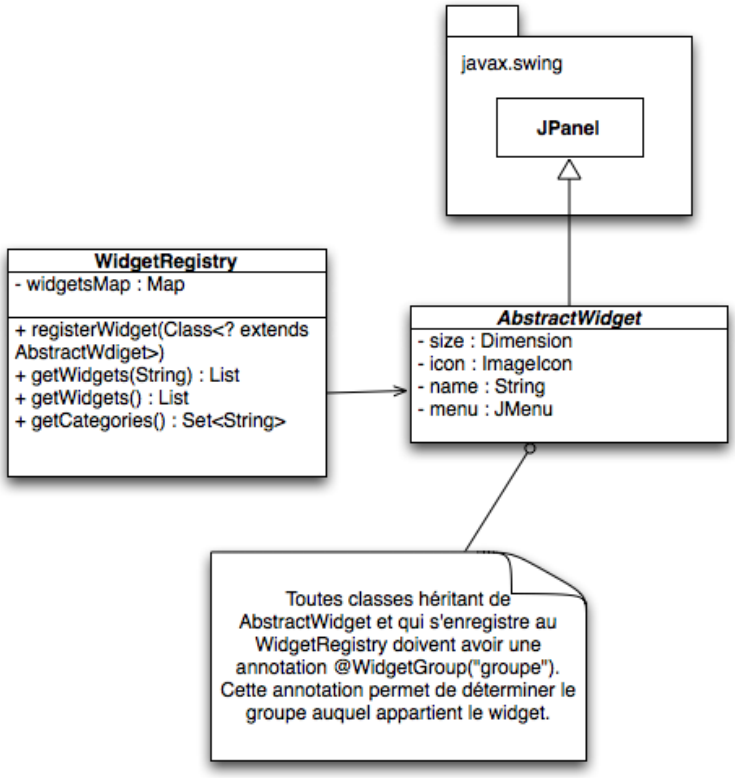


Figure 4-9— Diagramme de classe du sous-système de composants graphiques

Cette solution offre au développement l'isolation de chacun des composants graphique. En regroupant chaque composant graphique dans leur propre classe, la cohésion des composants est augmentée. Cependant, toute classe représentant un composant graphqiue doit être déclarée dans le WidgetRegistry afin que les mécanismes offerts par la télémétrie puissent le prendre en charge.

CHAPITRE 5

RÉSULTATS

5.1 Travaux réalisés

En excluant les artefacts exigés par le cours, voici la liste des artefacts réalisés durant le projet.

5.1.1 Document de vision (DE6)

Document décrivant la solution à être développé en terme de besoins et fonctionnalités du point de vue des intervenants.

Ce document a servi à cerner le projet entre les différentes parties prenantes.

5.1.2 Modèle du domaine (DE8)

Modélisation du domaine d'intérêt regroupant les entités, leurs attributs et les relations.

Ce document a servi à comprendre les interactions entre le système. Également, il a été utile afin de comprendre et d'illustrer l'architecture JAUS à partir de la norme AS5710.

5.1.3 Glossaire (DE9)

Document regroupant et expliquant les différents termes et acronymes techniques.

Ce document a permis à chacun des intervenants de s'entendre sur les termes du domaine.

5.1.4 Diagramme des cas d'utilisation (DE12)

Représentation graphique de l'interaction entre les cas d'utilisations et leurs acteurs.

Le diagramme des cas d'utilisation a permis de certains certains cas d'utilisation typiques. Il a permis également d'identifier les utilisateurs typiques.

5.1.5 Cas d'utilisation (DE13)

Document décrivant le comportement du système lorsque celui-ci répond à des requêtes

Les quelques cas d'utilisation ont permis de planifier le comportement du système espéré. Au cours du développement, il a été déterminer que les cas d'utilisation n'ont pas beaucoup de valeur pour ce genre de projet. Des scénarios de qualités logiciels ont été favorisés.

5.1.6 Maquettes fil de fer (DE14)

Représentations visuelles de la structure, de l'organisation, de la navigation et de l'interaction de l'interface graphique du système.

Au cours du projet quelques maquettes fil de fer ont été réalisé. Cependant, la portée du projet a été modifiée. La nouvelle portée réduisait la valeur des maquettes de fil de fer.

5.1.7 Cahier d'architecture (DE15)

Document qui capture, justifie et explique les décisions architecturales.

Ce document a été rédiger dans l'esprit qu'il soit utilisé dans le futur. Il a pour objectif d'expliquer les décisions qui ont été prises lors du développement du projet. Ceci a pour objectif d'éviter les réécritures d'application.

5.1.8 Scénarios de qualités logiciels (DE17)

Document qui expose des qualités logicielles désirées à l'aide de scénarios. Cet artefact a été ajouté en cours de projet.

En ajoutant des scénarios de qualités, il est possible d'évaluer la qualité du logiciel développé et aussi d'établir des objectifs clairs et pris durant le projet.

5.2 Résultats

Cette section présente chaque objectif établis au début du projet et de quelle façon l'objectif a été atteint.

5.2.1 Communication bidirectionnelle entre l'opérateur et le robot

Ce premier objectif consiste à permettre à l'opérateur et au robot de s'échanger des informations.

Dans sa forme actuelle, l'application permet à l'opération d'envoyer des commandes au robot cible. Également, elle permet à l'opérateur d'envoyer des commandes afin que le robot puisse interagir avec son environnement.

Cette image affiche la télémétrie en pleine action. Elle affiche le composant graphique permettant à l'utilisateur de visualiser la profondeur actuelle du robot à gauche. À la droite, elle affiche le composant graphique permettant d'envoyer des commandes de contrôle aux moteurs du robot.

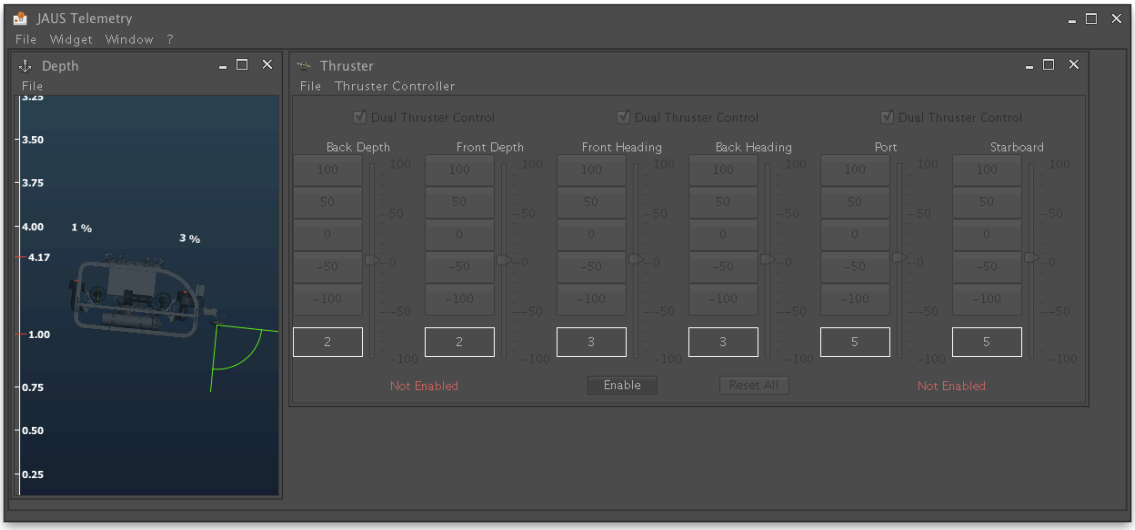


Figure 5-1— Communication bidirectionnelle entre l'opérateur et le robot

Comme la télémétrie assure une communication directionnement, il est raisonnable de conclure que cet objectif est atteint.

5.2.2 Compatibilité assurée entre plusieurs systèmes autonomes

Cet objectif doit assurer que le système de téléopération soit réutilisable sur plusieurs plateformes autonomes.

Depuis l'hiver 2011, la télémétrie est utilisée par les clubs Capra, Dronolab et S.O.N.I.A.. Chacun des regroupements bénéficie donc d'une base commune. Également, chacun des clubs a développé leurs propres composants graphiques afin de répondre à leur besoin.

La télémétrie est utilisée entre différents clubs de robotique qui ont des robots différents. L'objectif de compatibilité entre plusieurs systèmes autonomes est atteint.

5.2.3 Développement accéléré des composants graphiques

Cet objectif concernait l'ajout de fonctionnalité au système. L'application doit permettre aux développeurs d'ajouter facilement de nouveaux composants graphiques.

Lors des dernières semaines, plusieurs composants graphiques ont été développés. L'ajout d'un nouveau composant graphique implique :

- ajout de deux classes Java;
    - une classe pour la communication JAUS;
    - une classe pour la représentation graphique Swing;
- modification d'une classe Java permettant d'enregistrer la classe.

L'application contient également plusieurs mécanismes afin d'assister le développeur. Entre autres :

- encapsuler la configuration;
- faciliter la traduction de l'interface graphique;
- afficher les composants;
- gérer les dépendances.

Ainsi, le développement de chacun des composants est compartimenté. Le développement s'en trouve accéléré. En conséquence, l'objectif est atteint.

5.2.4 Configuration flexible des composants graphiques

Cet objectif visait à assurer que les composants graphiques étaient configurables.

Actuellement, des mécanismes qui permettent de lire les fichiers de configuration encapsulent les détails d'implémentation au développeur. Ainsi, il est facile pour le développeur d'utiliser les paramètres de configuration.

La manipulation des fichiers de configuration implique :

- l'ajout d'une ligne afin d'ajouter un fichier de configuration;
- l'ajout d'une ligne afin d'obtenir la valeur associée à une clef configuration.

Ces mécanismes facilitent grandement la tâche des développeurs. Les développeurs peuvent offrir une configuration flexible de leurs composants.

Ainsi, ces mécanismes facilitent grandement l'ajout de configuration des composants. L'objectif de rendre les composants graphiques flexibles est donc atteint.

5.2.5 Personnalisation simple de l'interface de téléopération

Le système doit permettre d'accommoder certaines personnalisations désirées par l'utilisateur. Cet objectif visait la personnalisation de l'interface.

Dans sa forme actuelle, l'interface permet quelques personnalisations telles que :

- la langue d'utilisation;
- le thème de couleur;
- sauvegarde et chargement du positionnement des composants de l'interface graphique.

Cette image affiche la liste des thèmes disponible.

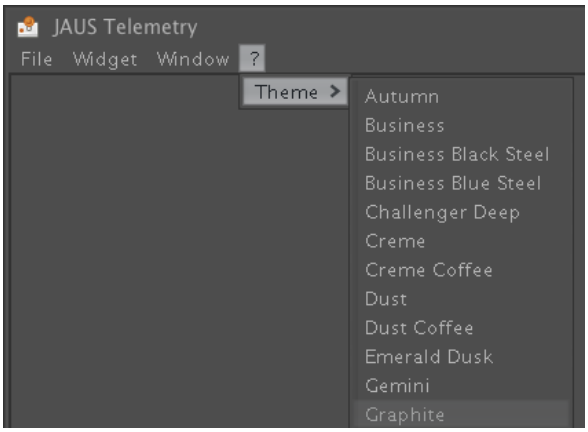


Figure 5-2 — Liste des thèmes disponibles

Les images suivantes affichent les capacités de sauvegarde et chargement de la disposition des composants graphiques. Également, elle affiche la même section d'interface dans deux langues différentes.

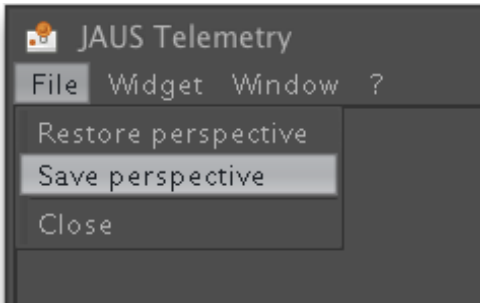


Figure 5-3 — Sauvegarde et chargement de la disposition de l'interface graphique en anglais

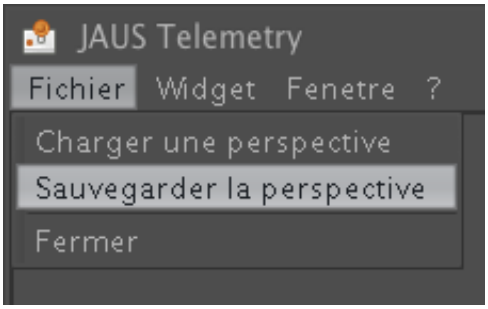


Figure 5-4 — Sauvegarde et chargement de la disposition de l'interface graphique en français

Il est donc possible de personnaliser l'interface de téléopération dans une certaine mesure. Ainsi, cet objectif est marqué comme accompli.

5.2.6 Développement de prototypes de composants graphique.

Cet objectif visait à développer quelques prototypes de composants graphiques afin d'évaluer les mécanismes offerts par le cadre d'application.

Le développement d'un seul composant a été menées dans le cadre de ce projet. Le composant de la console a permis de tester le cadriciel. L'image suivante affiche le composant graphique de console.

Cependant, plusieurs autres composants graphiques ont été développés à l'externe de ce projet. L'opinion des développeurs au sujet du développement de nouveaux composants graphiques a été recueillie afin d'établir la satisfaction des développeurs.

CONCLUSION

Au cours des dernières années, les différents clubs de robotiques ont développé leur propre solution de téléopération. Cette solution permet de communiquer avec le robot afin d'obtenir les données de son capteur et de lui demander d'effectuer certaines actions. Cependant, comme chacun des clubs développe leur solution, les efforts de développement sont considérablement augmentés malgré le fait qu'une majeure partie du problème est semblable entre chacun des clubs.

Le projet visait à jeter les bases d'un cadre applicatif afin de permettre à chacun des clubs de partager la charge de travail que représente une interface de téléopération. Ce cadriciel devait permettre à chacun des clubs d'étendre les fonctionnalités afin de répondre à leur besoin. Elle devait également permettre de fournir une base solide pour les besoins partagés par les clubs.

La solution développée dans le cadre de ce projet permet à chacun des clubs de partir d'une base commune solide. Elle fouine également des mécanismes facilitant le développement. De plus, elle permet à chacun des regroupements étudiants d'ajouter des fonctionnalités à la télémétrie afin de répondre à leurs besoins. Finalement, les utilisateurs pourront personnaliser leur interface lors des activités d'opérations.

RECOMMANDATIONS

Malgré les bonnes pratiques instaurées et le souci de vouloir bien faire les choses, plusieurs recommandations sont formulées dans cette section afin d'améliorer le projet.

D'abord, une première amélioration possible est de favoriser l'utilisation de système de contrôle de version décentralisé. En utilisant cette forme de système, il sera possible de séparer le développement en plusieurs endroits. Également, chaque développeur a une copie de l'historique de développement. Ceci permet le développement même si le développeur n'a pas de connexion avec le serveur distant.

Une deuxième amélioration possible est d'encourager l'esprit de revue des pairs. La revue des pairs est une pratique qui permet d'améliorer la qualité des logiciels construits. Elle permet à deux personnes de s'interroger si les approches utilisées sont les bonnes. Elle permet également de corriger les erreurs d'inattention.

Finalement, une troisième recommandation est d'extraire la partie interface graphique de l'application et de permettre sa réutilisation. Plusieurs applications du club S.O.N.I.A. sont basées sur les mêmes cadres applicatifs d'interface graphique. En généralisant le tout, il serait possible de réduire les efforts de développement de régler les problèmes beaucoup plus efficacement.

BIBLIOGRAPHIE

Bass, Len, Paul Clements et Rick Kazman. 2003. Software architecture in practice, 2nd. Coll. « SEI series in software engineering ». Boston: Addison-Wesley, xxii, 528 p.

Bloch, Joshua. 2008. Effective Java, 2nd. Upper Saddle River, N.J.: Addison-Wesley, xxi, 346 p. p. <http://www.loc.gov/catdir/toc/fy0805/2008926278.html>.

Cajolet-Laganière, Hélène, Pierre Collinge et Gérard Laganière. 1986. Rédaction technique et administrative, 2e éd. rev. et augm. Sherbrooke: Éditions Laganière, 331 p. p.

Clements, Paul. 2010. Documenting software architectures : views and beyond, 2nd. Upper Saddle River, N. J.: Addison-Wesley, xxxix, 537 p. p.

Gamma, Erich. et al. 1994. Design Patterns : Elements of Reusable Object-Oriented Software. Upper Saddle River, N.J.: Addison-Wesley, 416 p.

Haase, Chet, et Romain Guy. 2008. Filthy rich clients : developing animated and graphical effects for desktop Java applications. Coll. « The Java series ». Upper Saddle River, NJ: Addison-Wesley, xxvii, 572 p. p.
<http://www.loc.gov/catdir/toc/ecip0717/2007019818.html>.

Krug, Steve. 2000. Don't make me think! : a common sense approach to Web usability. Coll. « Circle.com library ». Indianapolis, Ind.: QUE, ix, 195 p. p.

Larman, Craig. 2005. Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development, 3rd. Upper Saddle River, N.J.: Prentice Hall PTR, xxv, 703 p. p.

Lauesen, Soren. 2005. User interface design : a software engineering perspective. Harlow, England ; New York: Pearson/Addison-Wesley, xviii, 604 p. p.

SAE. 2008. Aerospace Standard AS5710, JAUS Core Service Set.

Tidwell, Jenifer. 2006. Designing interfaces, 1st. Beijing ; Sebastopol, Calif.: O'Reilly, xx, 331 p. p. adfasdfasdfasdfasdf

ANNEXE I

DOCUMENT DE VISION

Suivi des changements

Numéro de version	Date	Numéro de figure, table ou section	A M S	Description
1.0	2011/01/17	1, 2, 3, 4, 5	A	Version initiale
1.1	2011/01/20	1, 2, 3, 4, 5	M	Version révisée par le client

1. Introduction

Ce document expose la problématique ainsi que la solution à développée. Il définit du point de vue des intervenants le produit à développer en terme de besoins et de fonctionnalités. Ce document pourra être utilisé par les nombreux intervenants afin d'obtenir une bonne compréhension du problème.

Ce document explique la problématique que le produit vient résoudre, expose la solution assurée par le produit, définit les différents intervenants et décrit l'environnement des utilisateurs.

2. Positionnement

2.1 Énoncé du problème

Le problème est	Lorsqu'une organisation développe un système interagissant avec son environnement, il est essentiel de savoir comment le système capte cet environnement. Il est également important de pouvoir envoyer des instructions au système. Afin d'accomplir ces objectifs, une interface de téléopération est développée par ces organisations afin de visualiser les informations reçues de la plateforme. Cette interface sert aussi à envoyer des instructions au robot. Cependant, une partie importance d'un tel système de communication bidirectionnelle est équivalente pour différentes plateformes.
affecte	• Les opérateurs de robots autonomes qui désirent utiliser le système de communication bidirectionnelle afin d'effectuer des tâches. • Les membres des différents clubs de robotiques qui désirent ajouter des fonctionnalités au système de communication. • Les observateurs du système autonome qui désire comprendre la situation actuelle du robot.
dont l'impact est	• Une duplication des travaux de développement et de configuration • Un effort de développement plus important • Une qualité réduite du produit développé • Une exécutions ralenti des opérations conduites par le robot. • Des problèmes de compréhension de la situation actuelle du robot.
une solution idéale serait	De disposer d'un système qui : • est une interface de communication bidirectionnelle avec un robot. • rassemble les besoins communs des organisations afin de fournir une base commune • est flexible afin de répondre aux besoins spécifiques des différents clubs.

	• répond aux besoins regroupement étudiant.

2.2 Positionnement du produit

Pour	Les membres du club scientifique S.O.N.I.A. et les autres clubs oeuvrant dans la robotique.
Qui	• désirent visualiser en temps réel la situation actuelle de robots • désirent envoyer des commandes au robot • désirent ajouter des fonctionnalités au système au système de communication • désirent utiliser une interface conviviale et personnalisable • désirent partager les efforts de développement • désirent produire des suites logiciels d'une plus grande qualité
La Telemetrie JAUS	est une solution de base d'échange d'information bidirectionnelle
offre	• une communication en temps réel entre le robot et son opérateur. • une interface d'utilisation intuitive et personnalisable. • un mécanisme de modulartié des composants graphiques. • le partage des efforts de développement.
Contrairement à	l'interface de télémétrie actuelle incompatible avec d'autres robots, peu flexible et difficilement compréhensible.
Notre produit	doit être compatible avec différentes plateformes robotiques, doit avoir une interface utilisateur intuitive, personnalisable et flexible.

3. Descriptions des intervenants

3.1 Sommaire des intervenants

Nom	Description	Responsabilités
I1. Capitaine de club étudiant	Ces personnes sont l'autorité technique et hiérarchique d'un regroupement étudiant. Ils s'assurent que les objectifs de l'année sont atteints.	Le capitaine de club étudiant doit : • garantir que le projet est rempli dans les délais prescrits. • s'assurer que le projet répond aux besoins des membres de l'équipe. • gérer les ressources. • supprime les entraves au développement.
I2. Responsable logiciel	Ce rôle représente l'autorité logicielle d'un groupe étudiant. Ils doivent s'assurer que les objectifs logiciels de l'année sont réalisés.	Le responsable logiciel est responsable : • de la maintenabilité de la solution. • de la flexibilité à long terme de la plateforme. • de la bonne intégration du système à l'intérieur de l'environnement. • coordonne la développement.
I3. Développeur logiciel	Cette entité représente un développeur logiciel. Il doit développer des solutions afin d'accomplir les objectifs techniques de son club scientifique.	Le développeur logiciel a comme responsabilité : • d'effectuer le développement de la solution. • utiliser la boîte à outils de développement.
I4. Opérateur	Cette responsabilité est partagée par toute personne qui sera appelée à utiliser l'application.	• L'opérateur doit faire usage du système en opérant la plateforme robotique.
I5. Vulgarisateur	Cette responsabilité englobe tout membres d'un regroupement étudiant qui doit expliquer une plateforme robotique au public dans le cadre d'activité de promotion	• Le vulgarisateur est tenu d'expliquer ce que le robot capte et de faire la correspondance avec ce que les

	tel que les portes ouvertes de l'ÉTS.	observateurs externes voient.

3.2 Environnement de l'utilisateur

Le club scientifique S.O.N.I.A. a comme objectif de construire une plateforme autonome. Ce système sert à participer à une compétition internationale organisée par l'AUVSI et ONR. Cet évènement se déroule annuellement à San Diego en Californie.

Les étudiants ont donc une année afin de construire un prototype. Leur robot doit être capable d'accomplir diverses épreuves complexes.Ces opérations simulent des tâches que les robots sous-marins autonomes déployés dans les eaux ont à effectuer.

Afin de tester leur sous-marin, le regroupement S.O.N.I.A. doit effectuer différents tests. Les tests les plus représentatifs de la compétition sont les tests effectués en eaux. Lors de ses tests, l'opérateur dispose d'une application nommée Télémétrie. Ce programme communique avec l'entité servant de centre de décision du robot.Cette application permet de voir les différentes valeurs captées ainsi que d'envoyer des commandes au sous-marin.

Cet environnement d'utilisation est peu convivial. Le temps de test en eaux est coûteux et limité. Également, tout moment perdu à comprendre la situation actuelle du sous-marin est du temps que le robot n'est pas en service.

Actuellement, la télémétrie actuelle communique avec le centre de décision à l'aide d'un protocole développé par S.O.N.I.A. Ce protocole est loin d'être optimal et efficace. Aucune correspondance n'existe dans l'industrie et donc n'est pas interopérable.

Plusieurs applications sont en cours de développement. Le centre de décision est en cours de réécriture afin d'intégrer l'architecture JAUS. Le projet de téléopération devra pouvoir communiquer avec la nouvelle architecture JAUS et n'aura pas besoin de communiquer avec l'ancien système. En transférant la suite logicielle actuelle vers une architecture JAUS, il

ainsi possible pour les clubs de bénéficier de solutions logicielles communes .

4. Vue d'ensemble du produit

4.1 Besoins et fonctionnalités du produit

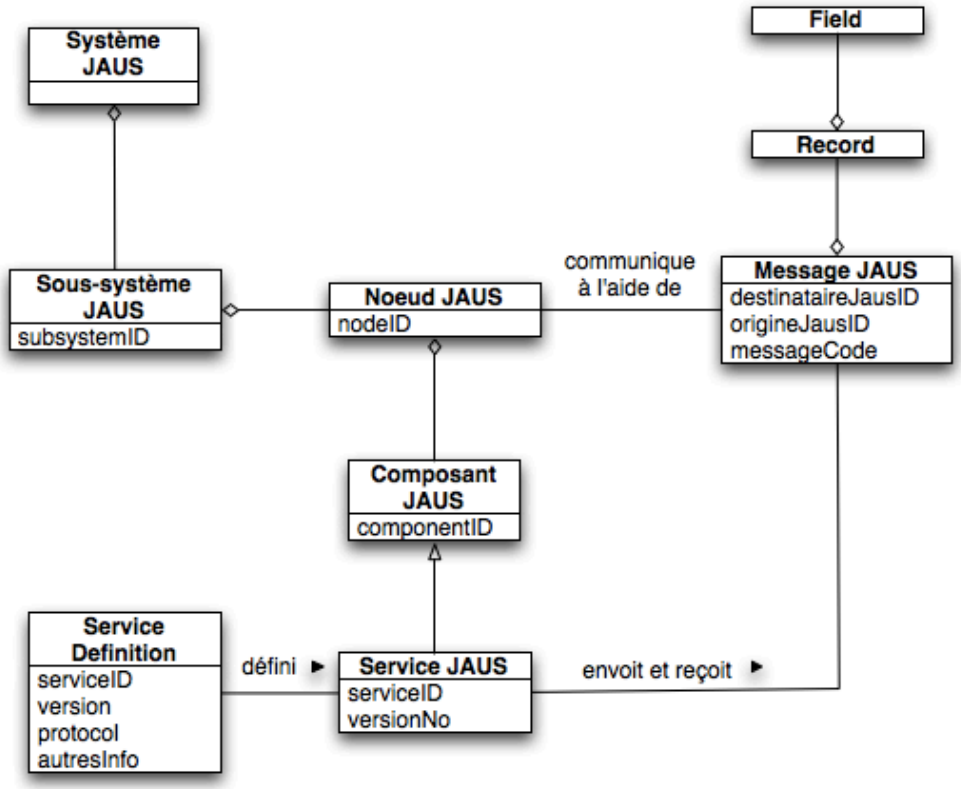
Besoin	Priorité	Solutions proposées
B1. L'interface de l'utilisateur devra être personnalisable	1	Un mécanisme de personnalisation sera implémenté.
B2. L'interface d'utilisation devra permettre le chargement et la sauvegarde de personnalisation.	2	Un mécanisme de sauvegarde et chargement sera implémenté.
B3. Les composants graphiques devront être réutilisables.	3	Les composants graphiques disposeront d'une interface de configuration.
B4. Le système devra être capable d'avoir de nouveaux composants graphiques.	4	Un mécanisme d'ajout de composants graphiques sera implémenté.
B5. Le système devra permettre le développement de nouveaux composants graphiques.	5	Une boîte à outils de développement de composants graphiques sera disponible.
B6. Les composants devront être facilement compréhensibles.	6	Des tests d'utilisation seront menés durant le développement.
B7. Le système devra afficher les données du véhicule autonome.	7	Des composants graphiques représentant l'état actuel du véhicule autonome seront développés.
B8. Le système devra envoyer des commandes au véhicule autonome.	8	Des composants graphiques permettant d'envoyer des instructions au véhicule autonome seront développés.

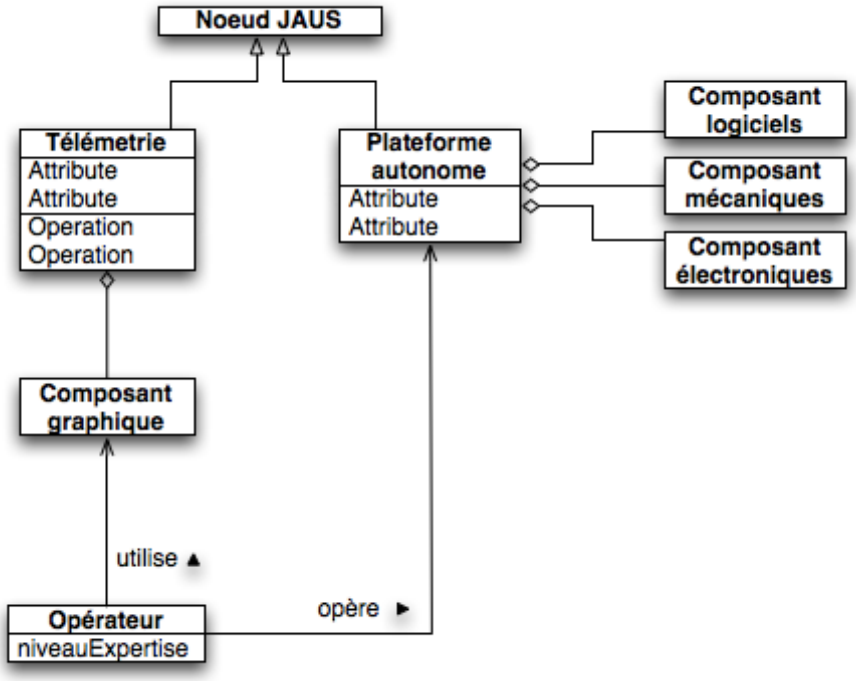
5. Exigences supplémentaires

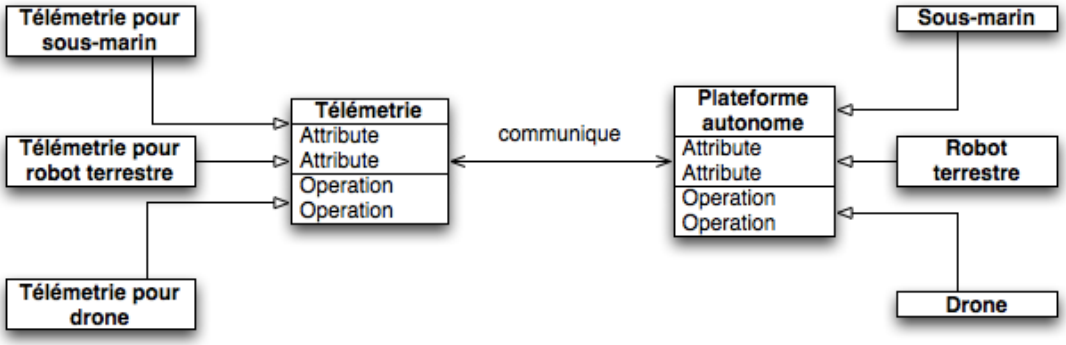
Exigence	Priorité
E1. Le système doit intégrer l'architecture JAUS.	1
E12 La solution doit être implémentée avec le langage de programmation Java.	2
E3. La solution doit être compatible avec Linux, Mac OS X et Windows.	2

ANNEXE II

MODÈLE DU DOMAINE




Noeud JAUS
Télémetrie
Attribute
Attribute
Operation
Operation
Plateforme autonome
Attribute
Attribute
Composant logiciels
Composant mécaniques
Composant électroniques
Composant graphique
utilise
Opérateur
niveauExpertise
opère


Télémetrie pour sous-marin
Télémetrie pour robot terrestre
Télémetrie pour drone
Télémetrie
Attribute
Attribute
Operation
Operation
communique
Plateforme autonome
Attribute
Attribute
Operation
Operation
Sous-marin
Robot terrestre
Drone

ANNEXE III

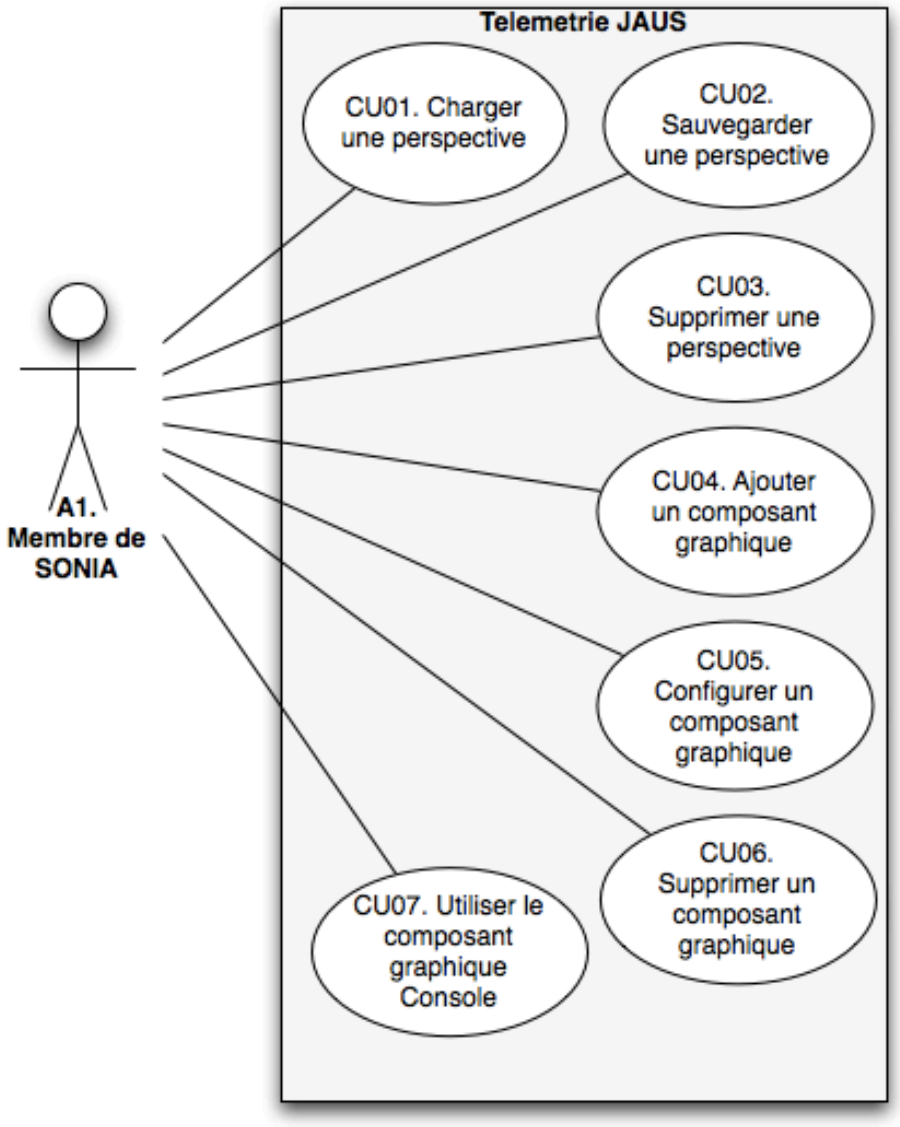
GLOSSAIRE

Terme	Définition
S.O.N.I.A.	Système d'opération nautique intelligent et autonome.
Club S.O.N.I.A.	Le club scientifique S.O.N.I.A. est une organisation regroupant des étudiants de l'ÉTS. Le regroupement a comme objectif de construire un robot sous-marin automne afin de participer à une compétition internationale de sous-marin autonome organisé par la fondation de l'AUVSI.
AUVSI	Association for unmanned vehicle systems international. L'association a pour objectif de promouvoir et supporter la communauté de système autonome. Elle a mis en place une fondation qui organise les compétitions de robots autonomes en Amérique du Nord.
ÉTS	École de technologie supérieure. École d'ingénierie située à Montréal.
ONR	Office of Naval Research. Organisation qui coordonne, exécute et promeut les programmes de science et technologie de la marine américaine.
JAUS	Joint Architecture for Unmanned Systems. Norme développée par SAE International. Elle définit une architecture ouverte dans le domaine des véhicules autonomes. Cette norme définit une structure hiérarchique d'élément et décrit un ensemble messages afin de supporter l'interopérabilité entre les systèmes autonomes. Les normes actuelles disponibles sont JAUS Core Service Set (AS5710A), JAUS Transport Standard (AS5669A) et JAUS JAUS Mobility Service Set (AS6009).
Système JAUS	Un système JAUS est l'entité qui contient des sous-systèmes JAUS. Dans le cadre de ce projet, le système JAUS est inexistant, car c'est ce qui regroupe l'architecture au complet.
Sous-système JAUS	Un sous-système JAUS représente une entité regroupant plusieurs noeuds JAUS. Dans le cadre de ce projet, un sous-système JAUS est créé pour chaque club. S.O.N.I.A. et Dronolab ont leurs propres sous-système JAUS.
Noeud JAUS	Un noeud JAUS est une entité indépendante dans l'architecture JAUS. Dans le cadre de ce projet, l'interface de téléopération et le système de contrôle du sous-marin sont deux noeuds JAUS.

Composant JAUS	Un composant JAUS est un élément logiciel à l'intérieur d'un Noeud JAUS. Dans le cadre du projet, un composant JAUS est associé à chaque capteur et actuateur. Un classe actuateur termine avec les mots «SensorComponent». Une classe représentant un senseur termine avec les mots «DriverComponent».
Service JAUS	Un service JAUS est l'entité qui reçoit les messages JAUS et envoie des messages JAUS. Il peut recevoir et envoyer des messages. C'est ça responsabilité a créer et interpréter les messages échangés. Dans le cadre de ce projet, chaque composant JAUS est associé à au moins un service JAUS. Il existe des services JAUS pour chaque composant. De plus, il y a présence de service JAUS supplémentaire afin de remplir les exigences de la norme.
Message JAUS	Un message JAUS est ce qui permet les échanges de données. Un message de catégorie report répond à un message de catégorie « query ». Également, un message de type command permet d'envoyer des messages de modification.
Record JAUS	Un record JAUS une entité qui regroupe les différentes valeurs contenues dans un message.

ANNEXE IV

DIAGRAMME DE CAS D'UTILISATION



ANNEXE V

CAS D'UTILISATIONS

Cas d'utilisation #01 : Charger une perspective existante

Description :

Ce cas d'utilisation décrit la séquence d'action qui permet de charger une perspective de l'interface de téléopération.

Acteur primaire :

Membre du club S.O.N.I.A.

Pré-condition :

L'application de téléopération est en cours d'exécution.

Postcondition :

La perspective en cours a été modifiée.

La perspective en cours est celle affichée par défaut.

Scénario normal :

Acteur	Système
1. L'utilisateur désire changer la perspective en cours.	
	2.Le système affiche la liste des perspectives disponible.
3. L'utilisateur consulte la liste des perspectives.	
4. L'utilisateur indique la perspective à utiliser.	
	5. Le système charge la nouvelle perspective.
	6. Le système sauvegarde le choix de l'utilisateur

	comme choix par défaut. 7. Le système affiche la nouvelle perspective sélectionnée.

Scénarios alternatifs :

Acteur	**Système**
4.a1 L'utilisateur indique l'ajout d'une perspective au système. 4.a3 L'utilisateur indique le fichier.	4.a2 Le système demande à l'utilisateur d'indiquer le fichier représentant la perspective désirée. 4.a4 Le système ajoute la perspective au système. 4.a5 Le cas d'utilisation reprend à l'étape 5.
4.a3a1 L'utilisateur n'indique pas de fichier ou indique un fichier invalide.	4.a3a2. Le système indique à l'utilisateur que son choix est invalide. 4.a3a3. Le cas d'utilisation reprend à l'étape 4.a2
4.a3b1 L'utilisateur annule l'opération. 4.a3b2 Le cas d'utilisation termine en échec.	

Acteur	**Système**
5.a2 L'utilisateur indique avoir pris connaissance de l'erreur	5.a1 Le système n'arrive pas à charger la perspective. 5.a2 Le système indique l'erreur à l'utilisateur. 5.a4 Le cas d'utilisation reprend à l'étape 4.
	5.b1 Le système n'arrive qu'a chargé la perspective partiellement. 5.b2 Le système indique l'erreur à l'utilisateur. 5.b3 Le cas d'utilisation reprend à l'étape 7.

Cas d'utilisation #02 : Sauvegarder une perspective

Description :

Ce cas d'utilisation décrit la séquence d'action qui permet de sauvegarder une perspective de l'interface de téléopération.

Acteur primaire :

Membre du club S.O.N.I.A.

Pré-condition :

L'application de téléopération est en cours d'exécution.

Postcondition :

La perspective en cours a été sauvegardée.

Scénario normal :

Acteur	**Système**
1. L'utilisateur indique qu'il désire sauvegarder la perspective en cours.	
	2. Le système demande à l'utilisateur s'il désire sauvegarder la perspective à l'intérieur du système ou sous forme de fichier.
3. L'utilisateur indique qu'il désire sauvegarder la perspective à l'intérieur du système.	
	4. Le système demande un identifiant de perspective.
5. L'utilisateur indique l'identifiant désiré.	
	6. Le système sauvegarde la perspective à l'intérieur du système.

Scénarios alternatifs :

Acteur	**Système**
3.a1 L'utilisateur indique qu'il désire sauvegarder la perspective sous forme de fichier. 4.a3 L'utilisateur indique le fichier.	3.a2 Le système demande à l'utilisateur le fichier à sauvegarder. 4.a4 Le système sauvegarde la perspective à l'intérieur du fichier.
4.a3a1 L'utilisateur annule l'opération. 5.a3a1 Le cas d'utilisation termine en échec.	
4.a4a3 L'utilisateur indique avoir pris connaissance de l'erreur. 4.a4a4 Le cas d'utilisation reprend à l'étape 4.a3	4.a4a1 Le système obtient une erreur lors de la tentative de sauvegarde du fichier. 4.a4a2 Le système indique à l'utilisateur l'erreur.

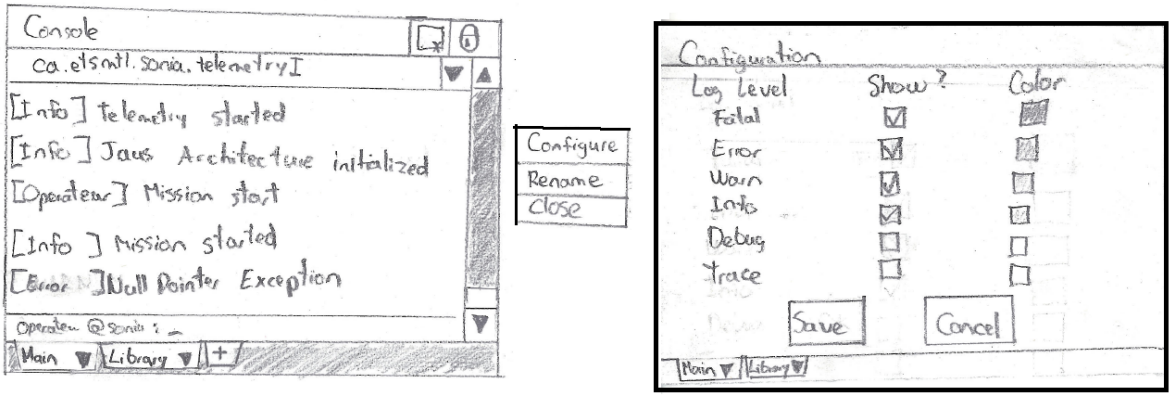
Acteur	**Système**
5.a1 L'utilisateur indique un identifiant invalide ou déjà utilisé. 5.a3 L'utilisateur indique avoir pris connaissance de l'erreur.	5.a2 Le système indique à l'utilisateur que l'identifiant est invalide ou est déjà utilisés. 5.a4 Le cas d'utilisation reprend à l'étape 5.
5.b1 L'utilisateur annule l'opération.	

5.b2 Le cas d’utilisation termine en échec.	

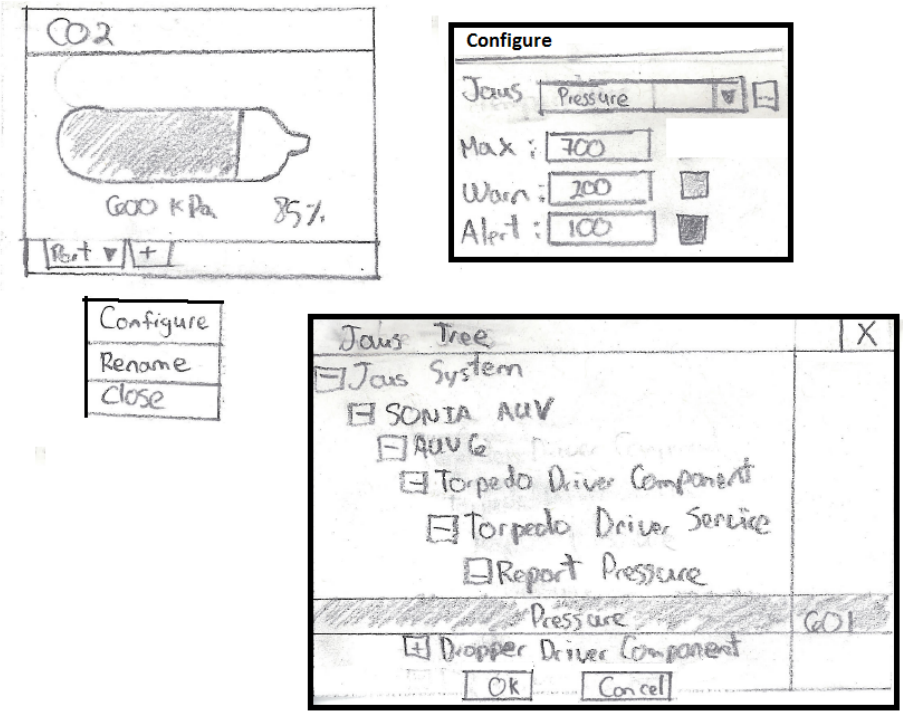
ANNEXE VI

MAQUETTES FIL DE FER

Maquette de la console



Maquette du composant graphique affichant l'état du CO2


CO2
600 kPa
85%
Port
+
Configure
Rename
Close
Configure
Jaus
Pressure
Max : 700
Warn : 200
Alert : 100
Jaus Tree
X
Jaus System
SONIA AUV
AUV 6
Torpedo Driver Component
Torpedo Driver Service
Report Pressure
Pressure
601
Dropper Driver Component
OK
Cancel

ANNEXE VII

SCÉNARIOS DE QUALITÉ

Scénario de qualité 1

Attribut de qualité : modificabilité

Élément	Valeur générique	Valeur spécifique
Source	Développeur	Développeur d'un club scientifique.
Stimuli	Désire ajouter une fonctionnalité	Désire ajouter un composant graphique.
Artefact	Interface graphique du système	Sur l'interface graphique du système.
Environnement	Au temps de développement	Au temps de développpement.
Réponse	Localiser les emplacement dans l'architecture à être modifié	Les changements à l'application sont localisé à un seul endroit.
Mesure de la réponse	Coût en terme de nombre d'élément à être modifié.	5 minutes de développement 1 module.

Un développeur d'un club scientifique étudiant désire développer un nouveau composant graphique sur l'interface graphique du système. En excluant le coût de développer le composant graphique, le temps d'intégration du composant au système pour qu'il soit utilisable doit prendre 5 minutes et doit localisé dans un module.

Scénario de qualité 2

Attribut de qualité : convivialité

Élément	Valeur générique	Valeur spécifique
Source	Utilisateur	Opérateur d'une plateforme autonome
Stimuli	Adapter le système	Personnaliser son système
Artefact	Système	Système
Environnement	Au temps d'exécution	Au temps d'exécution
Réponse	Interface personalisable	Personnalisation des composants de l'interface graphique
Mesure de la réponse	Satisfaction de l'utilisateur	90% des utilisateurs hautement satisfait de l'interface

Un opérateur d'une plateforme autonome désire personnaliser le système afin que ce dernier soit convivial et réponde à ses besoins. Les composants de l'interface graphique doivent être hautement personnalisables.Ces personnalisations doivent permettre à 75% des utilisateurs d'être satisfait.

ANNEXE VIII

CAHIER D'ARCHITECTURE

1. Objectif

Ce document est un cahier d'architecture concernant le système « Télémétrie Jaus ». Le cahier d'architecture sert à regrouper la philosophie, les décisions, les contraintes, les justifications, les éléments significatifs et tout autre aspect du système qui ont forgé la conception et l'implémentation de la solution.

Essentiellement, l'objectif de cet artefact est de permettre à l'équipe de comprendre comment le système est partitionné et organisé. Il permet aussi de donner un aperçu du système et d'exposer les motivations techniques aux futurs responsables du système.

2. Objectifs de l'architecture et philosophie

L'objectif principal de l'architecture du système est de construire un squelette logiciel qui répond aux besoins de plusieurs clubs de robotiques. Ce squelette fournit des services de base communs à chacun des clubs de robotique. Un exemple typique du genre de service offert est une couche de communication JAUS ou des mécanismes de gestion de configuration.

La philosophie principale du système est de bâtir un logiciel solide et extensible. Le système doit être solide, car le système doit répondre aux besoins de chacun des clubs de robotiques. Essentiellement, en construisant une base solide, chacun des clubs pourra bénéficier des avantages du partage de connaissances en toute quiétude.

Également, en permettant au club d'étendre les fonctionnalités, chacun des clubs sera responsable de leurs propres fonctionnalités. Il est donc essentiel que chaque composant commun soit bien isolé, que les modifications aient peu d'impact et que ses composants

coeurs fournissent des services fiables. De plus, il est à noter que plusieurs modules du logiciel évolueront en parallèle grâce aux efforts combinés de chacun des clubs.

3. Hypothèses et dépendances

Notre système est bâti sur plusieurs hypothèses et est bâti sur plusieurs dépendances d'éléments clefs. Cette section énumère ces éléments.

D'abord, les principales hypothèses sont :

- Les usagers ont de bonnes intentions.

En plus, le projet repose sur les dépendances suivantes :

- Une libraire permettant d'échanger des messages entre systèmes autonomes en utilisant l'architecture JAUS existe et est sans problème.

4. Requis de l'architecture

Cette section du document réfère aux scénarios de qualités présentés dans l'artefact DE17.

5. Décisions, contraintes et justifications

5.1 Architecture JAUS

L'architecture JAUS a été retenue afin d'assurer la communication entre les différents composants logiciels.

L'architecture JAUS est une architecture reconnue dans le domaine de systèmes autonomes. Cette norme sanctionnée par «The Society of Automotive Engineer » (SAE) est reconnue par l'AUVSI. L'AUVSI est l'entité qui organise les différentes compétitions étudiantes de véhicules autonomes. Le standard est reconnu dans diverses industries telles que la défense et l'industrie aérospatiale. Également, ce standard est utilisé dans plusieurs secteurs académiques.

Une autre raison d'utiliser l'architecture JAUS est que l'une des tâches à réaliser lors de la compétition est d'avoir un véhicule autonome pouvant interpréter et répondre aux messages JAUS. Ainsi, en basant notre système sur JAUS la compatibilité est assurée. Donc, pour la compétition l'équipe peut se concentrer à réaliser les tâches autonomes plutôt que de se préoccuper de la compatibilité avec JAUS durant les quelques jours de la compétition.

Finalement, l'objectif principal du projet est de bâtir une infrastructure commune entre les différents clubs de robotique autonome. L'architecture JAUS propose une solution compatible entre plusieurs véhicules autonomes. Il s'agit donc d'une solution idéale afin de bâtir une solution logicielle commune entre les différents clubs de l'ÉTS.

5.2 Plateforme Java

La plateforme Java a été retenue afin d'implémenter la solution logicielle.

D'abord, la majorité des systèmes logiciels développés par le club S.O.N.I.A. sont bâtis sur la plateforme Java. Le regroupement étudiant possède un vaste écosystème bâti autour de cette plateforme. Également, les membres du club ont déjà une expérience importante autour de Java.

De plus, les clubs étudiants de l'ÉTS sont essentiellement composés d'étudiants de l'université. L'école offre une formation autour de l'environnement Java. Une hypothèse raisonnable à établir est qu'un nouveau membre de club qui ne connait pas le langage Java aura la chance d'apprendre durant de sa formation.

Un autre aspect de l'utilisation de la plateforme Java est que celui-ci est du paradigme orienté objet. Ce paradigme promeut la réutilisation de code et facilite la maintenabilité du projet à long terme. Il permet également de mieux séparer les responsabilités en créant des abstractions. Ces abstractions serviront comme bloc de construction afin de construire le

système. Encore une fois, les notions avancées du paradigme orienté-objet font partie du contenu de la formation enseigné à l'ÉTS.

Le langage est portable entre les différentes plateformes. Il s'agit donc du langage de choix pour assurer la compatibilité entre le déploiement du véhicule et du développement. La majorité des développeurs à SONIA utilisent le système d'exploitation Windows. Certains développeurs utilisent le système Mac OS X. Cependant, le sous-marin auquel le système sera déployé fonctionnera sous le système d'exploitation Linux. Également, la distribution Linux peut varier selon les clubs ou les années de déploiement.

Le langage Java est reconnu pour être orienté objet et simple. Le langage Java contient peu de mécanismes qui pourraient obscurcir le code et réduire la compréhension. De cette façon, on assure la maintenabilité du code. Les nouveaux membres pourront comprendre facilement comment le système est bâti lorsqu'ils seront familiers avec le paradigme orienté objet. Java offre plusieurs mécanismes automatiques venant assister les développeurs. Par exemple, le mécanisme de « ramasse-miettes » vient réduire les risques de fuite mémoire. Java offre également des mécanismes de haut niveau permettant de faire de la concurrence. Ces mécanismes permettent surtout de s'attarder sur la construction de la solution plutôt que s'occuper des détails de bas niveau.

Finalement, il ne faut pas négliger que Java possède une bibliothèque applicative riche afin de construire nos applications. L'API de base possède un nombre élevé de méthode et de classe à utiliser. De plus, de nombreuses autres bibliothèques applicatives de haute qualité non officielle sont disponibles gratuitement. La communauté Java est très riche et permet de bâtir des applications sur les épaules des géants.

Bref, le langage Java est une plateforme de choix afin de développer la solution Télémétrie JAUS.

5.3 Swing

Afin de développer l'interface graphique de la Télémétrie Jaus, la trousse à outils Swing a été utilisée.

Une des raisons principales pour laquelle la boite à outils Swing a été utilisée est qu'elle est fournie par Java. Ainsi, il n'est pas nécessaire d'inclure une nouvelle bibliothèque applicative.

La télémétrie est composée de plusieurs éléments d'affichage complexe. Ces éléments sont plus pertinents à être traité et affiché du côté client afin d'enlever la charge du serveur. Ainsi, nous avons choisi une technologie de client lourd avec Swing.

En plus, Swing est une technologie connue par le regroupement SONIA. Le club possède déjà plusieurs applications Swing. Une hypothèse raisonnable est que le club a une déjà une expérience importante dans le développement d'application avec Swing.

5.4 Gestion de dépendances

Afin de gérer les différentes dépendances, les projets, le choix s'est arrêté sur le système de gestion de dépendance Maven.

Maven permet de simplifier la gestion des libraires. Ainsi, l'utilisation de Maven nous permet d'éviter le téléchargement manuel d'une vingtaine de dépendances. La gestion de ces dépendances externes est assurée par Maven. Également, Maven nous permet de nous assurer que les projets dépendent d'une version de fichier spécifique.

Également, Maven nous permet de simplifier les dépendances au projet externe. Nous pouvons interchanger fichiers jar et localisation du code source. Ceci nous permet donc de faire abstraction de la façon que les dépendances entre les projets sont gérées.

Bref, Maven nous permet de faciliter la gestion des dépendances pendant nos activités de développement.

5.5 Spécialisation des modules

Afin de répondre aux différents critères de réutilisation pour chacun des clubs, une conception favorisant les spécialisations des composants a été utilisée.

Cette conception permet de développer des modules généraux qui sont partagés par les clubs. Ces modules sont robustes et éprouvés dans différents environnements. Cette conception permet également à chacun des clubs de spécialiser ces composants pour leur besoin. Finalement, cette conception permet de séparer les responsabilités organisationnelles. Ainsi, le club SONIA ne sera pas responsable des éléments qui sont la responsabilité des autres clubs.

5.5 Internationalisation

Afin de pallier à plusieurs situations, l'application supportera l'internationalisation de son interface graphique.

D'abord, l'internationalisation rend l'interface plus agréable à utiliser. Cette fonctionnalité permet à chacun d'utiliser l'application dans la langue désirée. De plus, l'expérience de l'utilisateur est grandement améliorée lorsque celui-ci peut choisir sa langue de travail. Ainsi, la convivialité est améliorée.

Un autre argument supportant la mise en place d'un mécanisme d'internationalisation est de permettre aux clubs de faire des présentations dans la langue désirée. Comme les clubs sont appelés à faire des présentations au Québec et à l'étranger, il est adéquat de permettre le changement de langue selon le contexte.

Finalement, en supportant l'internationalisation de départ, ceci facilite le développement dès le départ. Ainsi, le patrimoine de code existant n'aura pas à être réusiné afin de supporter les capacités d'internationalisation.

5.6 Sérialisation personnalisée

Une décision importante a été d'utiliser un format personnalisé de sérialisation. Afin de pouvoir restaurer les objets dans leur état, nous avons utilisé le format YAML.

Cette approche à deux avantages. D'abord, ceci permet de contrôler le format de données sérialisé. Ainsi, en contrôlant ce format, il est possible de sérialiser seulement ce qui est nécessaire. Il est également possible, lors de la restauration de la sérialisation de restaurer les éléments essentiels. Ceci favorise donc une approche itérative lors du développement des capacités de sérialisation.

D'un autre côté, le format YAML est un format simple. Ce format est en texte clair ce qui permet donc à l'usager de comprendre et de modifier facilement les paramètres. Ceci rend donc les fichiers facilement interchangeables. Également, lors d'ajout des paramètres, il est possible pour les usagers d'ajouter et de supprimer les éléments modifiés sans avoir à refaire leur configuration complètement.

6. Mécanisme d'architecture

6.1 Spécialisation des composants

Un mécanisme utilisé est la spécialisation des composants. La spécialisation des composants permet de créer un groupe de composants générique qui peut être utilisé par tout un regroupement. Ensuite, il y a la création d'un nouveau regroupement séparé auquel on ajoute de nouveaux composants à ce groupe.

6.2 Encapsulation des informations

Afin de réduire le couplage, il est important d'encapsuler les informations. En encapsulant au maximum les objets, il est possible de créer des composants simples et isolés qui n'ont qu'une seule responsabilité. Cette pratique permet de réduire le couplage, augmenter la

cohésion et d'isoler les modules. Des modules isolés permettent de faciliter la testabilité et augmentent la maintenabilité du système.

6.3 Injection de dépendances

L'injection de dépendance est une approche qui permet de faire de l'inversion de contrôle. On injecte les dépendances d'un service à l'aide du constructeur. Ainsi, celui-ci ne connait pas l'implémentation et peut utiliser l'interface sans se soucier de l'implémentation. Cette pratique réduit le couplage et augmente la maintenabilité des composants.

6.4 Encapsulation des bibliothèques applicatives externes

Une autre pratique reconnue est l'encapsulation des bibliothèques applicatives externes. Cette pratique est appelée variation protégée et est popularisée par Craig Larman. Ainsi, on réduit la vulnérabilité au changement apporté par les mainteneurs de la libraire. Cette pratique réduit également le couplage aux bibliothèques externes, car leur point d'utilisation est localisé à un nombre endroits minimal.

7. Abstractions principales

7.1 Projet octets-commons

Le projet octets-commons regroupe toutes classes qui pourraient être utiles à plusieurs applications. Ce projet contient des abstractions telles que

- des classes facilitant les activités de création de fichier de journalisation;
- des classes d'interface graphique partagée par les applications;
- des classes utilitaires pour la réflexion;
- des classes pour assurer la gestion des configurations;
- des classes assistant la gestion de l'internationalisation.

Toutes ces classes peuvent être utilisées par chacune des applications supportées par les développeurs d'Octets. Cette base est utile afin de créer des applications robustes et de ne pas

à avoir a réinventer la roue pour chacune des applications développer. Ceci facile également la maintenance, car les modules sont réutilisés par de nombreux systèmes.

7.2 Projet jaus-library

Le projet jaus-library est l'abstraction principale de l'architecture JAUS. Il s'agit d'une implémentation de la norme AS5710 (« Core ») ainsi que de la norme AS6009 (« Mobility »).

La partie « Core » regroupe les mécanismes coeur de JAUS tels que la découverte, le contrôle d'accès et le transport d'information. La section « Mobility » regroupe les différents mécanismes permettant d'envoyer des commandes de déplacement au système autonome.

Le projet encapsule les mécanismes des communications JAUS. Elle contient les entités et les mécanismes nécessaires au fonctionnement de l'architecture JAUS.

Également, ce projet contient les messages que les systèmes autonomes utilisent afin de communiquer entre eux.

7.3 Projet jaus-telemetry

Le projet jaus-telemetry est le système qui permet à l'opérateur de communiquer avec un véhicule autonome.

Le projet permet à l'opérateur d'avoir accès à des composants graphiques qui communiqueront avec le système autonome pour lui indiquer de procéder à une opération spécifique. L'entité permet également d'afficher des informations provenant du système autonome à l'opérateur.

Il est important de noter que cette interaction est possible grâce à l'architecture JAUS implémentée.

7.4 Projet jaus-guinea-pig

Le projet jaus-guinea-pig est un système JAUS totalement virtuel. Il permet au développeur de tester l'architecture JAUS en obtenant des données.

Son utilité principale est de permettre les tests d'intégration du système. Il sert également lors du développement de composant graphique.

7.5 Projet sonia-jaus-library

Le projet sonia-jaus-library est une extension du projet jaus-library. Il s'agit du projet jaus-library auquel les membres de l'équipe S.O.N.I.A. ont ajouté ses propres messages afin d'obtenir les données nécessaires.

Les messages ajoutés sont essentiellement les actuateurs et les capteurs spécifiques au sous-marin S.O.N.I.A. tels que le lance-torpille ou l'écran LCD.

7.6 Projet sonia-jaus-telemetry

Le projet sonia-jaus-telemtry est une extension du projet jaus-telemtry. Il s'agit du projet jaus-telemtry auquel les membres de l'équipe S.O.N.I.A. ont ajouté leurs propres composants graphiques.

Il faut donc mentionner que ce projet repose sur la sonia-jaus-library afin d'envoyer les messages spécifiques au club S.O.N.I.A.

7.7 Projet sonia-jaus-guinea-pig

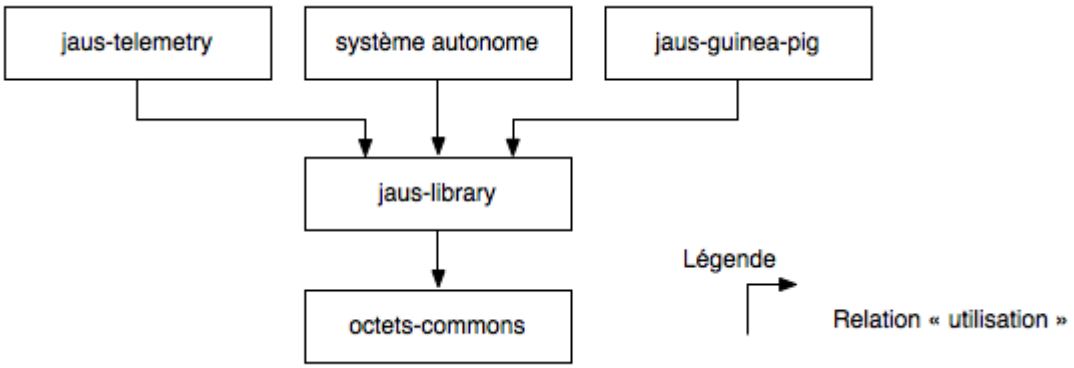
Le projet sonia-jaus-guinea-pig est une extension du projet jaus-guinea-pig. Il s'agit du projet jaus-guinea-pig auquel les membres sont ajoutés l'envoyé des messages contenu dans la libraire sonia-jaus-library. Ainsi, il est possible de tester l'envoi et la réception de messages spécifiques au club S.O.N.I.A.

8. Patrons d’architecture

Les patrons architecturaux sont expliqués dans les légendes. Il est important de noter que les vues sont basées sur les diagrammes en couche typiquement utilisés comme vue architecturale.

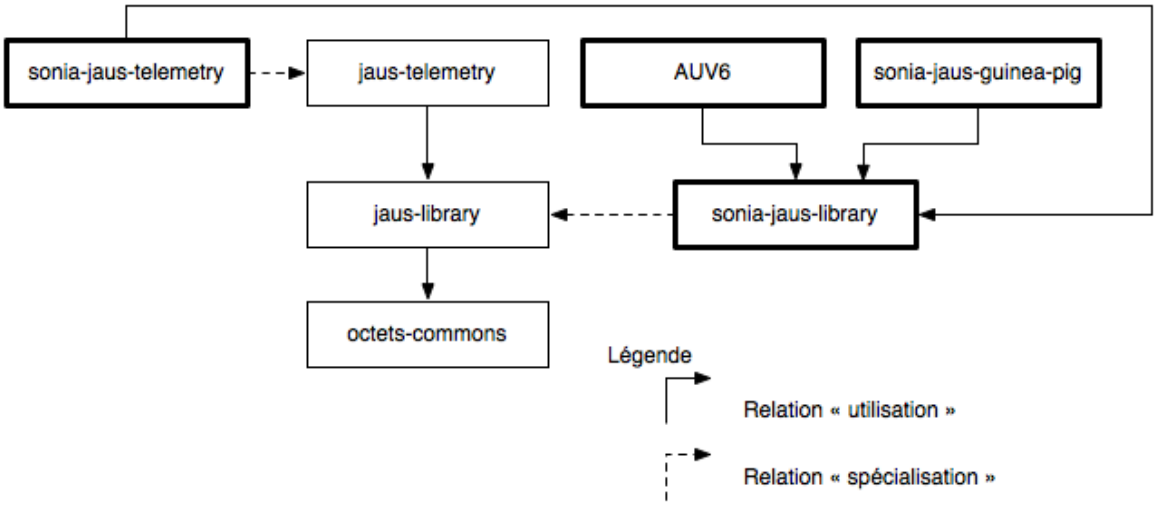
9. Vues architecturales

9.1 Architecture générique



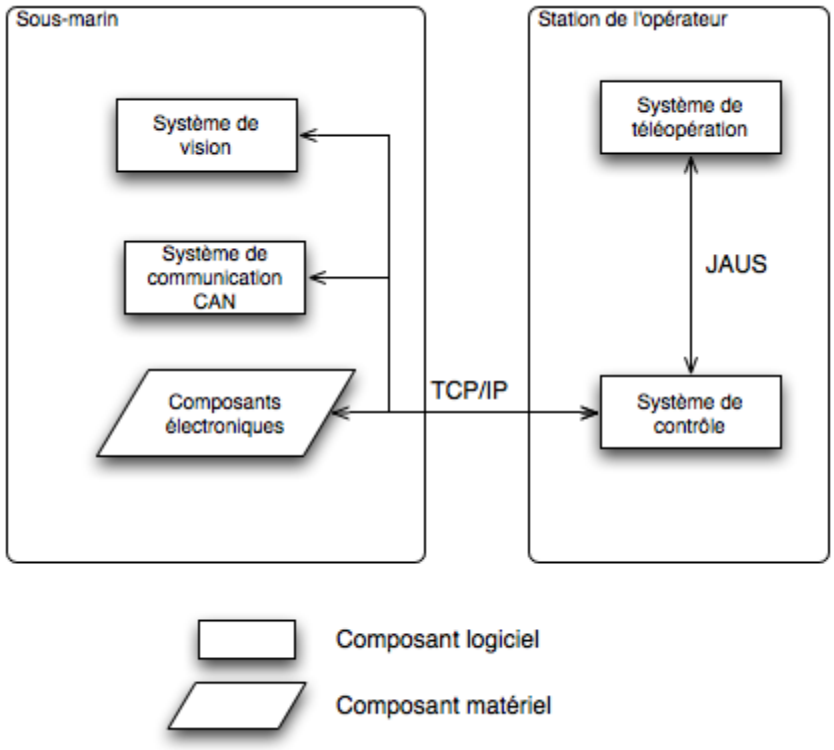
Cette vue indique les relations entre les différentes abstractions principales. Le projet octets-commons offre une base commune pour les logiciels. De plus, tous les systèmes jaus utilisent la jaus-library afin d'utiliser l'architecture JAUS et envoyer les courriels. Comme la jaus-library contient les informations afin d'utiliser l'infrastructure JAUS telle que les messages JAUS.

9.2 Architecture spécialisée à S.O.N.I.A.



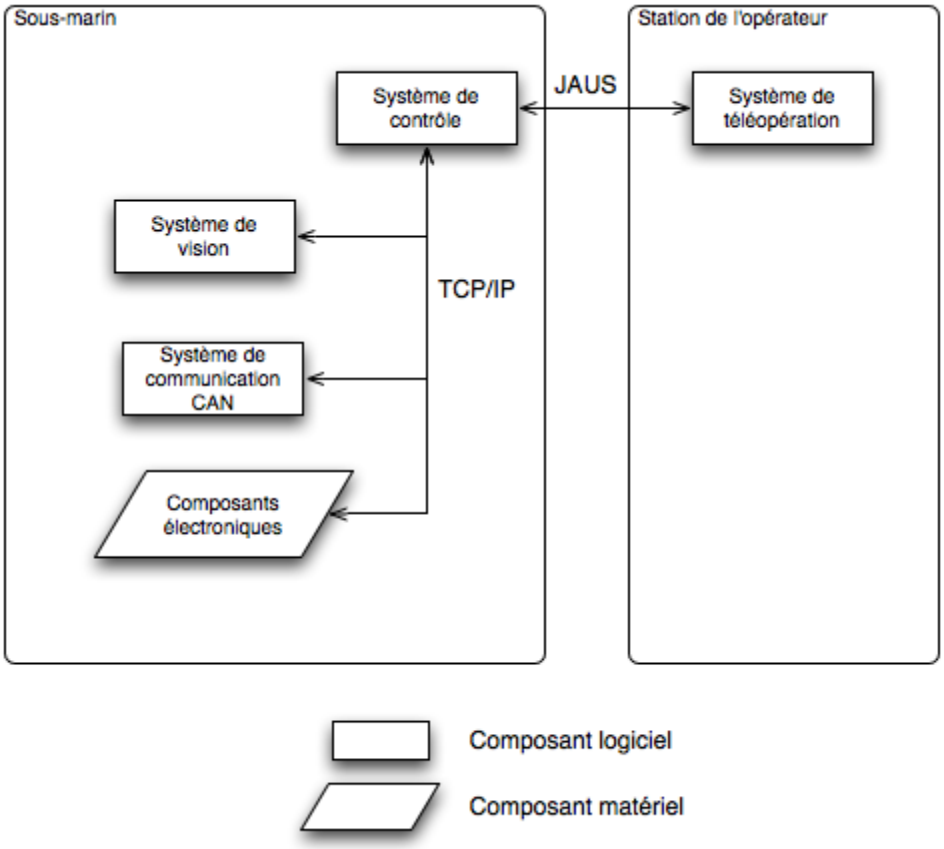
Cette vue architecture représente la solution spécialisée par S.O.N.I.A. La sonia-jaus-library vient spécialiser la jaus-library en ajoutant des messages JAUS supplémentaires. La sonia-jaus-telemetry vient spécialiser la jaus-telemetry en ajoutant des composants graphiques supplémentaires spécifiques à l'utilisation par le club S.O.N.I.A.

9.3 Déploiement du système en développement



Cette vue représente le déploiement du système lors du développement. L'utilisation du système de contrôle sur le poste de l'opérateur permet de faciliter la tâche de tester les systèmes. Il est plus convivial d'utiliser cette façon de déploiement lors du développement et des tests du système.

9.4 Déploiement du système pour exécution autonome



Cette vue représente le déploiement du système lors des opérations autonomes. Ainsi, si nécessaire, il est possible de couper la communication entre le système de contrôle et la télémétrie afin d'assurer le contrôle autonome du système.

ANNEXE IX

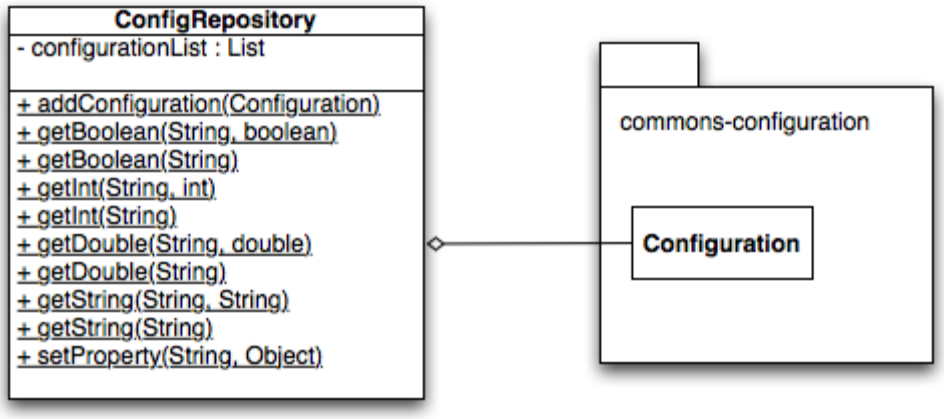
DOCUMENT DE CONCEPTION

1. Structure

La structure du système est documentée dans le cahier d'architecture.

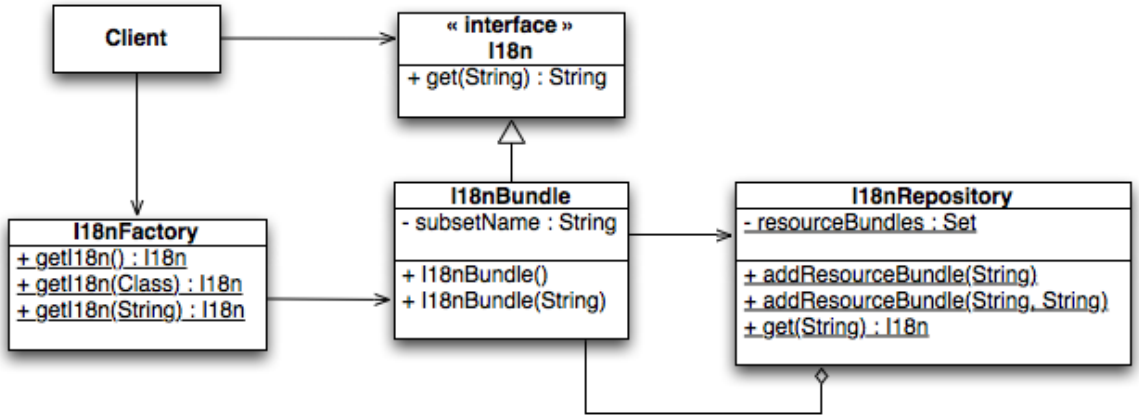
2. Sous-systèmes

2.1 Configuration



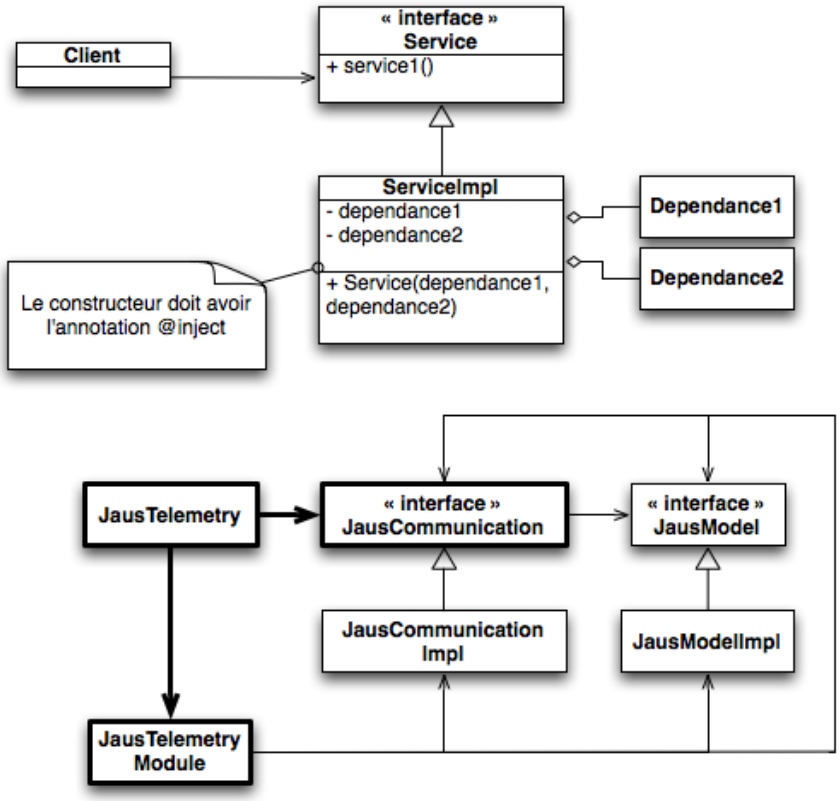
Cette conception permet d'encapsuler l'utilisation de la bibliothèque applicative commons-configuration. Également, l'accès aux configurations est localisé à un seul endroit. Les méthodes permettent d'obtenir tous les types de configuration possible. Ainsi, il est possible de profiter d'une interface de gestion de configuration conviviale pour le programmeur.

2.2 Internationalisation



Cette conception permet d'encapsuler l'utilisation des composants d'internationalisation. Grâce à cette conception, il est possible d'obtenir les valeurs des clefs facilement. Ces valeurs sont localisées par l'utilisation de l'interface i18n. Ainsi, la classe cliente n'est couplée qu'à deux classes.

2.3 Injection de dépendance

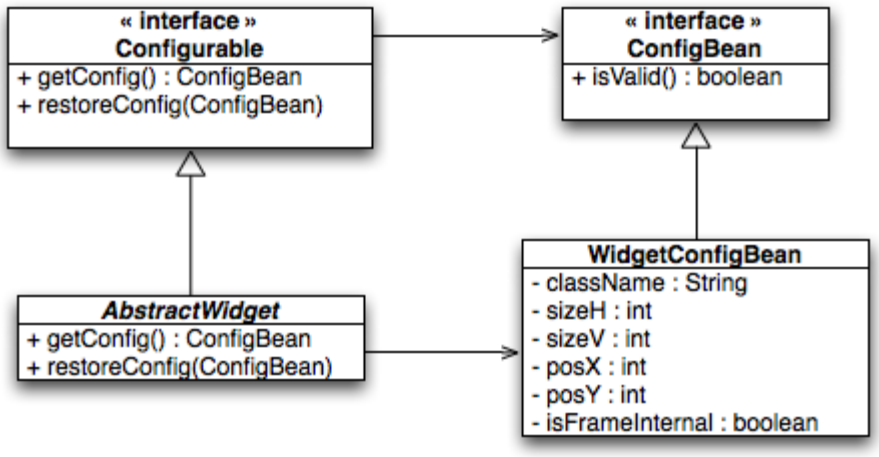


L'injection de dépendance permet de faire de l'inversion de contrôle. Cette inversion de contrôle permet de réduire le couplage entre les classes. Les classes clientes n'ont besoin que de connaitre l'interface qui implémente le service. De plus, les services sont automatiquement résolus par la libraire Guice.

Cet exemple illustre le faible couplage que procure l'injection de dépendances avec Guice. Comme l'illustre cet exemple, la classe cliente JausTelemetry n'est seulement couplé qu'à l'interface JausCommunication. Malgré que la classe JausCommunication soit couplée au JausModel, la classe JausTelemetry n'est pas couplée à celle-ci. Finalement, aucune classe

n'est couplée à une l'implémentation particulière d'une classe. Les classes sont seulement couplées à l'interface.

2.4 Sauvegarde de la disposition des composants graphiques

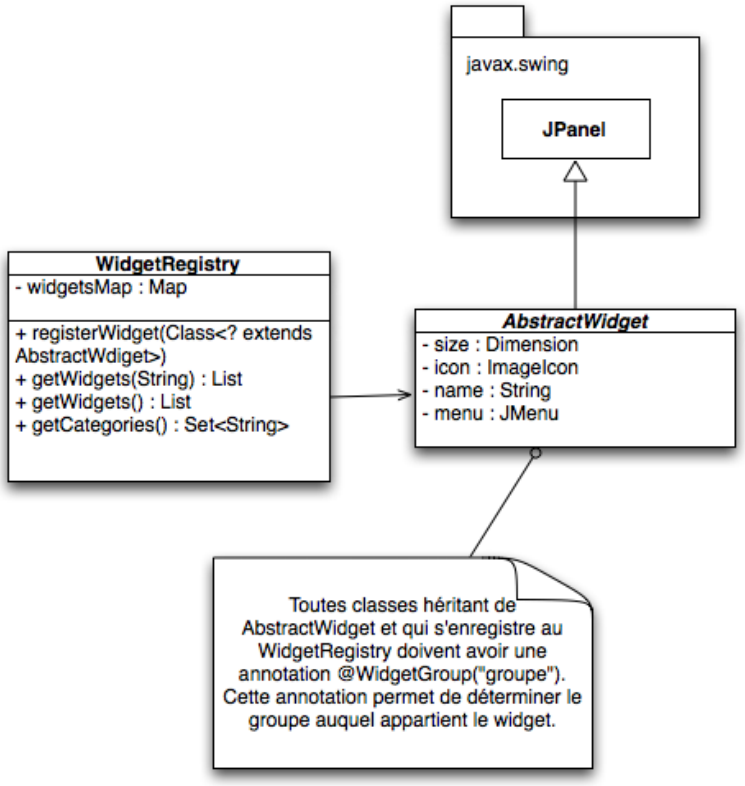


Cette conception fait appel au patron Memento. Lorsque la méthode getConfig() d'une instance de la classe AbstractWidget, celle-ci crée une instance de WidgetConfigBean représentant la configuration. La méthode restoreConfig permet de restaurer l'état à partir d'un bean. Cette configuration est ensuite sérialisée afin d'assurer la persistance des données.

Grâce à cette conception, lorsque l'on veut ajouter la possibilité de sérialisation à un composant il suffit d'implémenter l'interface configuration et de créer une classe qui implémentera la ConfigBean afin de représenter la configuration.

Grâce à cette conception, le composant graphique est isolé de sa configuration. Le couplage s'en retrouve réduit et permet ainsi de réduire le couplage.

2.5 Composant graphique



La conception illustrée représente la conception qui permet d'isoler les classes de composants graphiques des autres classes. Ainsi, afin de créer un nouveau composant graphique n'a qu'à implémenter la classe AbstractWidget et à ajouter l'annotation @WidgetGroup('groupe"). De plus, afin que le composant graphique soit ajouté à l'interface graphique il faut enregistrer la classe afin que celui-ci soit accessible.

Cette conception permet d'isoler les composants graphiques des autres composants et des sous-systèmes de la solution de téléopération. Ainsi, afin de créer un nouveau composant, il suffit de créer deux nouvelles classes et de modifier une autre classe.