THESE

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE

DE MÉCANIQUE ET D'AÉROTECHNIQUE

(Diplôme National — Arrêté du 25 mai 2016)

École Doctorale : Science et Ingénierie pour l'Information, Mathématiques (S2IM)
Secteur de Recherche : INFORMATIQUE ET APPLICATIONS

Présentée par :

Olga GOUBALI

****

APPORT DES TECHNIQUES DE PROGRAMMATION PAR DEMONSTRATION DANS UNE DÉMARCHE DE GÉNÉRATION AUTOMATIQUE D'APPLICATIFS DE CONTRÔLE-COMMANDE

****

Directeur de Thèse : **Patrick GIRARD**
Co-encadrants : Laurent GUITTET, Pascal BERRUET et Alain BIGNON
****

Soutenue le 30 Janvier 2017

devant la Commission d'Examen

JURY

Rapporteurs :	**Jean VANDERDONCKT**	Professeur, Université Catholique de Louvain
	Gaëlle CALVARY	Professeur, Université Grenoble-INP
Examinateurs :	**Christophe KOLSKI**	Professeur, Université de Valencienne et du Hainaut-Cambrésis
	Jean-François PÉTIN	Professeur, Université de Lorraine
	Patrick GIRARD	Professeur, Université de Poitiers, Poitiers
	Laurent GUITTET	Maître de Conférences, ISAE-ENSMA, Poitiers
	Pascal BERRUET	Professeur, Université de Bretagne-Sud, Lorient
	Alain BIGNON	Responsable R&I, SEGULA

À mon petit frère
Christian GOUBALI

Remerciements

Je me disais que les remerciements c'est la partie la plus facile à écrire mais je me rends compte maintenant que c'est quand même très difficile de trouver les mots pour exprimer toute ma gratitude envers chacun de vous, avec une pensée particulière.

Je tiens à remonter à l'origine de ces travaux, pour remercier tout particulièrement Alain BIGNON, sans lequel cette thèse n'aurait pas eu lieu. Tu étais là avant même le début, en me proposant de continuer en thèse après mon stage. L'envie de saisir cette opportunité a émergé des discussions enrichissantes qu'on a pu avoir et du fait que j'ai eu à participer à la construction de mon sujet. Le mot « Merci » est petit pour exprimer ma reconnaissance pour tes précieux conseils, pour ton soutien, pour le temps que tu as consacré à encadrer cette thèse.

Je remercie également mes autres encadrants de thèse : Patrick GIRARD, Laurent GUITTET et Pascal BERRUET d'avoir accepté d'encadrer cette thèse. Patrick, tes précieux conseils ont enrichi ces trois années de travail. Laurent, ton dynamisme et ta joie de vivre sont communicatifs ; merci pour tes conseils, ton aide et surtout ta disponibilité. Pascal, merci d'avoir eu la patience de composer avec des points de vue parfois un peu trop IHM et d'avoir apporté une vision beaucoup plus système, pour enrichir ces travaux.

Je souhaite remercier Messieurs Jean VANDERDONCKT et Christophe KOLSKI pour l'intérêt que vous avez porté à mes travaux lors de nos différents échanges. Monsieur VANDERDONCKT, merci de m'avoir faire l'honneur et le plaisir de proposer d'être rapporteur de cette thèse. Je remercie Madame Gaëlle CALVARY qui m'a également fait l'honneur d'accepté de relire cette thèse, ainsi que Messieurs Jean-François PETIN et Christophe KOLSKI pour avoir accepté de l'examiner.

J'adresse mes sincères remerciements à tout le personnel et doctorants (passé et présent) du LIAS. Les moments passés avec vous ont été très bénéfiques pour moi, tant sur le plan personnel que professionnel. Un grand merci à Claudine RAULT et Emmanuel GROLLEAU pour avoir toujours été présente quand j'en avais besoin. Je remercie tout particulièrement Thomas LACHAUME, Guillaume PHAVORIN, Nassima BENAMMAR et Anh Toan BUI LONG d'avoir rendu agréable, mes séjours à Chassneuil du poitou. Je n'oublie pas Selma BOUARAR qui a partagé avec moi des moments inoubliables.

J'étends mes remerciements à tous les collègues de SEGULA TECHNOLOGIE avec lesquels j'ai passé ces années de thèse. Merci à Perine Le SENECHAL et Éric LE BRIS pour l'intérêt que vous avez toujours bien voulu me porter, pour votre soutien, votre aide et vos conseils ; un grand merci à Perine pour avoir relu mes chapitres. Je souhaite remercier toute l'équipe Recherche et Innovation (Djamal KESRAOUI, Soraya KESRAOUI, Sophie PRAT et Fabien Silone) de Lorient pour l'expérience enrichissante et pleine d'intérêt qu'il m'a fait vivre durant cette thèse ; Djamal, merci d'avoir participé activement à la réalisation de mes tests utilisateurs. Merci à ma chère amie Soraya qui m'a toujours prêtée une oreille attentive. Je ne saurai terminer cette vague de remerciements sans penser à Laurianne BOULHIC et Line POINEL qui ont activement participé à la réalisation et à la retranscription des tests utilisateur ; un

grand merci à Laurianne pour l'analyse des résultats, la rédaction de l'évaluation et pour son soutien. Vous avez tous été adorables avec moi.

Comment puis-je oublier combien adorable sont avec moi, Laurent et Laurence GUITTET pour m'avoir accueilli plus d'une fois dans leur foyer. Grâce à vous des moments de solitude ce sont envolés. Merci à vous.

Merci à mes amis (qu'ils soient de longues dates ou pas), votre soutien a été précieux pour moi et les moments que nous passons ensemble sont, pour moi, autant de bons moments associés à cette thèse ; alors merci à : Aminath BADAROU, Lenaïg THÉPAULT-BIGNON, Nicolas AUFFRET, Lucresse et Nasser BADA, Nina AKPOVI... Un grand merci à tous ceux que j'ai pu mentionner car la liste est longue.

D'autres personnes m'ont permis de me lancer dans cette thèse. Tout d'abord, cette aventure en France n'aurait jamais débuté sans mon petit frère, Christian GOUBALI, qui a cru en moi, m'a soutenu tout au long de ces années et a tout donné pour que j'y arrive, merci. Je remercie beaucoup mes parents (Grégoire GOUBALI et Véronique BADEDJI) pour m'avoir donné l'opportunité de faire des études dans de bonnes conditions et pour m'avoir toujours soutenue. Je remercie également M. Charles GAGNON pour m'avoir encouragé à accepter de faire une thèse, m'avoir soutenu et aussi pour ses précieux conseils.

Cependant, se lancer dans une thèse est une chose, arriver jusqu'au bout en est une autre. Ces années de thèse n'auraient sans doute pas eu la même saveur sans l'équilibre et le soutien apportés par une personne très particulière : Davy RODIER, tu as partagé ma mauvaise humeur, mes problèmes, mon stress et tu es toujours présent quand j'en ai besoin, merci.

Pour me changer les idées, j'ai toujours pu compter sur toute la famille RODIER (Francine, René, Sébastien, Audrey et Michaël, sans oublier le petit dernier : Aiden). Je remercie particulièrement Francine d'avoir lu et corrigé mes chapitres de thèse. Ce n'est déjà pas évident pour quelqu'un du domaine mais toi tu as accepté de le faire sans hésiter. « Tu es une mère pour moi ».

Albert Pine, un écrivain du XIXe siècle a dit : « *Ce que nous faisons pour nous même disparaît avec nous. Ce que nous faisons pour les autres et le monde est immortel et demeure* ». Je tiens à vous dire que ce que vous avez tous fait pour moi restera marqué à jamais dans mon cœur. Grâce à vous j'ai pu aboutir. Merci à tous…

Cette thèse est le fruit d'une collaboration entre l'entreprise SEGULA Technologies, acteur majeur des métiers de l'ingénierie, le Lab-STICC, pôle de référence en recherche sur les systèmes communicants et le LIAS, pôle de référence en recherche sur l'ingénierie des données et modèles, les systèmes embarqués temps réels, l'automatique et système..

	SEGULA Technologies B. P. 50256 56602 Lanester Cedex http://www.segula.fr/fr
	Laboratoire d'Informatique et d'Automatique pour les systèmes Université de Poitier ISAE/ENSMA 1 Avenue Clément Ader, 86961 Chasseneuil du Poitou. http://www.lias-lab.fr/
	Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) - UMR n° 6285 Université de Bretagne-Sud Centre de recherche Christiaan Huygens, BP 92116 56321 Lorient cedex, France www.lab-sticc.fr

"How well we communicate is determined not by how well we say things, but how well we are understood."

Andrew Grove (1936 - 2016)

Sommaire

Liste des Figures

Liste des tableaux

Introduction générale

Les avancées technologiques ont rendu, depuis les années 1930, les systèmes industriels de plus en plus complexes. L'accroissement de cette complexité conduit à les considérer comme des systèmes sociotechniques[1] (Rauffet et al. 2013) car les composantes humaines, techniques et organisationnelles sont étroitement liées.

Lors de la conception de ces systèmes, les acteurs engagés dans le processus de conception mettent en jeu des compétences, des logiques et des points de vue qui sont différents, contradictoires, complémentaires et souvent paradoxaux car ils appartiennent à des corps de métiers différents. L'organisation des travaux de conception de tels systèmes suit généralement un processus séquentiel. Depuis les années 80, le cycle en V s'est imposé comme le standard des processus de conception dans l'industrie. La nature séquentielle du cycle en V rend coûteuses (en temps comme en argent) les modifications tardives pouvant intervenir lors de la conception d'un système. D'autres modèles d'organisation du travail comme les méthodes agiles prennent le contre-pied des approches traditionnelles pour proposer un processus de développement itératif et incrémental mais elles s'appliquent difficilement à des équipes de projet trop importantes et pluridisciplinaires.

Dans le contexte de projets de grande envergure comme la conception de systèmes sociotechniques complexes, le nombre des intervenants et la diversité des domaines d'études entraînent des problèmes de cohérence d'ensemble et d'interprétation des spécifications. De fait, les erreurs issues de ces problèmes de compréhension ne sont le plus souvent détectées que lors des phases d'essai du système. Selon une enquête réalisée par (Standish Group 2009), 32% des projets ont abouti (livrés à temps avec le respect du budget, des caractéristiques et fonctions exigées) ; 44% se sont terminés difficilement (en retard, avec dépassement du budget, sans toutes les caractéristiques et fonctions demandées) ; et 24% ont échoué (annulés avant la fin ou livrés sans jamais être utilisés).

Le contrôle-commande des systèmes sociotechniques complexes illustre à merveille la nature pluridisciplinaire des activités de conception. Ces travaux mettent, en effet, en œuvre des connaissances approfondies du système à contrôler, mais nécessitent également de l'expertise quant au système de contrôle commande (automatisme et informatique). Par ailleurs, la prise en compte du facteur humain nécessite la mise en œuvre de connaissances en ergonomie.

Le contrôle-commande a connu, depuis les années 70, d'importantes évolutions dues à la progression des technologies électroniques et informatiques (Fanet 1997). La Figure 1 présente la structure d'un système de contrôle-commande composé d'IHM de supervision permettant de surveiller et de commander le procédé, d'automates permettant de traiter des données (signaux électriques, demande d'exécution, renvoi des états et des alarmes etc.), d'actionneurs permettant de transformer des signaux électriques en grandeurs physiques et de capteurs permettant de transformer les grandeurs physiques en signaux électriques.

Plusieurs approches fondées sur le paradigme de modèle, telles que la Model Based System Engineering ont tenté d'apporter une solution aux problèmes liés à la conception des systèmes de contrôle-commande. L'Object Management Group propose une première formalisation des concepts de l'Ingénierie Dirigée par les Modèles (IDM) au travers de sa recommandation

[1] Système en forte interaction avec l'humain, comme par exemple, un navire.

MDA[2] (OMG 2003). Cette approche vise à élever le niveau d'abstraction des activités de programmation en plaçant les modèles au tout début du processus de développement logiciel. En complément d'autres paradigmes (objets, patrons…), l'IDM permet l'utilisation de modèles métiers maîtrisés par les experts du système à contrôler pour générer automatiquement des applications de contrôle commande (Bignon 2012).

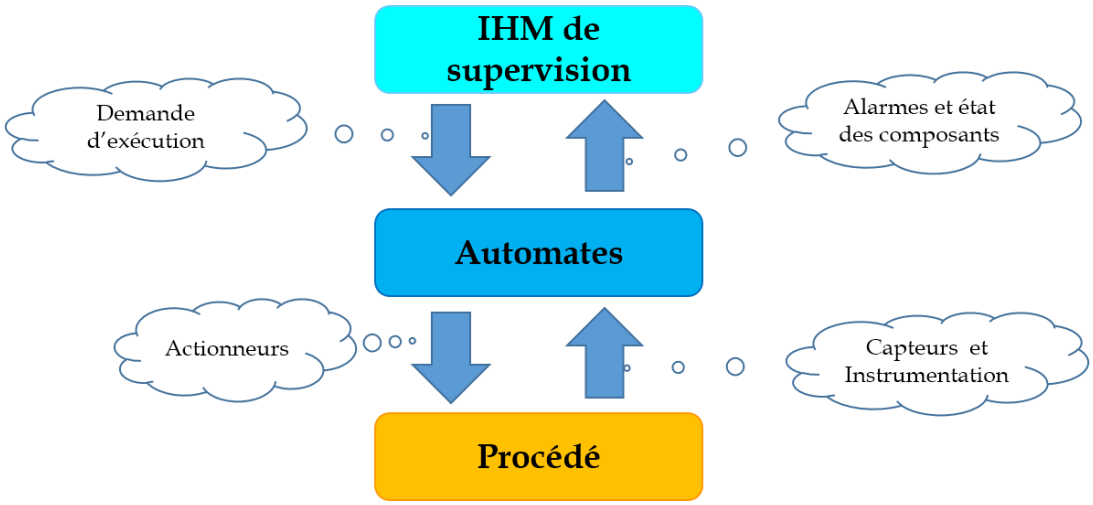


Figure 1 Structure du contrôle-commande

Bien qu'efficaces, les approches à base de modèles nécessitent un effort important de formalisation des données d'entrée (Arzapalo 2014). Or, une partie des connaissances nécessaires à la conception de systèmes complexes sont difficilement formalisables. Concernant la partie fonctionnelle du système à concevoir, des formalismes ont pu être proposés, mais ils sont finalement peu mis en œuvre car non maîtrisés par les acteurs de la conception.

Sur le plan fonctionnel, le niveau d'automatisme dans les systèmes de contrôle-commande a beaucoup évolué depuis quelques années (Fanet 1997). En effet, la répartition des tâches confiées, d'une part à l'opérateur, et d'autre part à l'automate, a beaucoup varié. De plus en plus de tâches sont automatisées (par exemple, la commande de plusieurs éléments du système avec un simple clic) à travers des commandes de haut-niveau plus proches du but final de l'opérateur, ce qui facilite le pilotage du système. En fait, les **commandes de haut-niveau** permettent d'automatiser la réalisation d'un ensemble de commandes élémentaires, ce qui permet d'exécuter plus facilement les fonctions. L'obtention des commandes de haut-niveau nécessite la spécification du fonctionnement du système sous forme de modèles d'entrée pouvant être utilisés par les approches basées sur l'IDM.

Les spécifications fonctionnelles décrivent la façon dont un système permet d'atteindre les buts de l'utilisateur et contiennent notamment une liste des principales fonctions du système et des scénarios d'exploitation. Elles contiennent les suites d'actions de l'opérateur sur le système, nécessaires à la réalisation d'une fonction avec prise en compte de toutes les possibilités (configurations). La définition des spécifications est à la charge de l'expert métier, qui les écrit typiquement en langage naturel, puis les communique aux concepteurs de

[2] Model Driven Architecture

l'interface de supervision et du programme de commande qui sont en charge de les implémenter et de les intégrer au système.

En complément des spécifications fonctionnelles, d'autres modèles comme les *modèles de tâches* ont démontré leur intérêt pour recueillir certaines exigences concernant l'utilisation des systèmes. En effet, les modèles de tâches permettent de décrire les actions à réaliser par un superviseur pour lancer les fonctions en utilisant les commandes de haut-niveau. La complexité et la répétitivité des tâches de supervision rendent néanmoins difficile la conception de ces modèles. Les modèles de tâches sont généralement décrits par les concepteurs d'IHM avec des outils de modélisation dédiés et peuvent être validés avec les experts métiers. La vérification et la validation des modèles de tâches s'effectuent par l'utilisation de simulateurs proposés dans les outils de modélisation de tâches. La plupart de ces simulateurs ne facilitent cependant pas la communication entre les utilisateurs et les concepteurs d'IHM (Lachaume et al. 2012).

Les communications entre les différents corps de métiers sont illustrées par la Figure 2. Les erreurs qui découlent de l'interprétation des spécifications sont dues à la différence de culture technique entre le prescripteur (expert métier) et l'exploitant (automaticien et/ou informaticien). Ces erreurs affectent le coût de développement et d'évolution des applications.

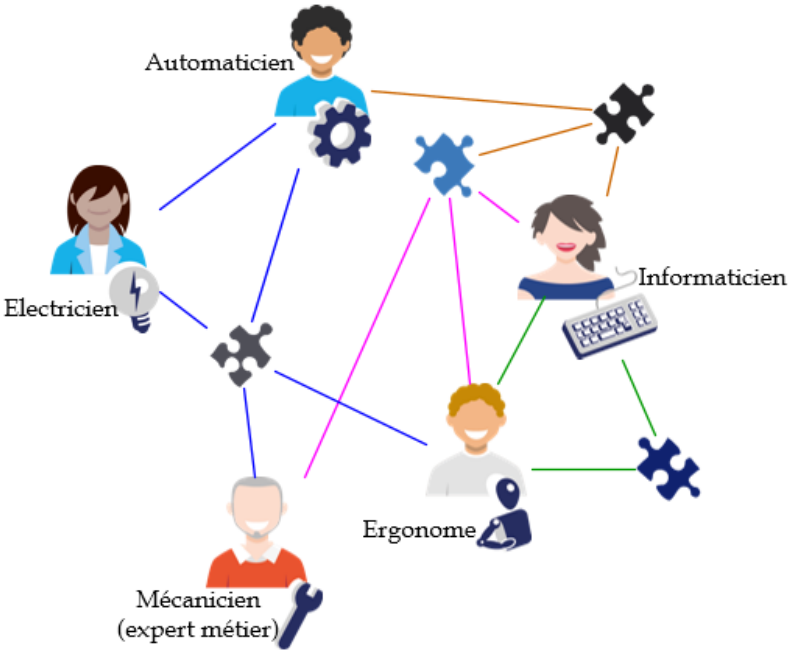


Figure 2 Communication entre les différents corps de métiers intervenant dans la conception d'un système de contrôle-commande

De plus, suivant la complexité du système, la conception des modèles de tâches et la définition des spécifications fonctionnelles peuvent être fastidieuses alors qu'elles sont essentielles pour la conception. Une étude publiée en 2014 par (Standish Group 2014) montre que les échecs des projets sont dus à 29.4% aux spécifications et exigences incomplètes, aux changements dans les spécifications et aux objectifs imprécis ; et à 12,8% au manque d'implication des utilisateurs finaux. Ces données reste inchangées depuis plus d'une vingtaine d'années car une étude publiée par le même groupe en 1995 (Standish Group 1995) avait conduit au même résultat. Il semble alors intéressant de proposer une démarche de spécification permettant de capturer au

mieux la connaissance de l'expert métier (utilisateur expert) pour obtenir plus facilement les modèles de tâches et les spécifications fonctionnelles, s'intégrant dans une démarche globale d'IDM. Notre objectif consiste ainsi à faciliter la communication entre les différents intervenants du projet, à permettre à chaque intervenant de se concentrer sur son corps de métiers et à donner le pouvoir à l'expert métier d'exprimer plus facilement les spécifications de son système afin d'avoir des spécifications précises et complètes.

Afin de valider notre démarche, nous l'appliquons à un cas concret du domaine naval. Le cas d'application considéré est un système de production, de distribution et de stockage de l'eau douce embarqué sur un navire de taille moyenne comportant une centaine de passagers (Figure 3), noté EdS[3] (Annexe 1). Ce système est vu comme un système sociotechnique car l'équipage est en forte interaction avec ce dernier. Il est automatisé et possède également des interfaces de supervision.

Le système EdS alimente le navire en eau douce et en eau déminéralisée pour les moteurs. Les fonctions principales du système sont : Distribution, Distribution à quai, Brassage, Embarquement, Débarquement, Production, Transfert. Ces fonctions sont appelées les fonctions de haut-niveau du système, et sont réalisées manuellement et/ou automatiquement. Le système EdS est reconfigurable (73 configurations unitaires à définir pour les 7 fonctions) de par le choix des cuves et des moyens de production et de pompage. Les 73 configurations ne tiennent pas encore compte de la possibilité d'effectuer plusieurs fonctions simultanément, charge à l'expert métier de définir ces 73 configurations ainsi que les cas d'exécutions simultanées. Généralement, les spécifications fonctionnelles (configurations écrites dans un fichier Excel et description en langage naturel) sont ensuite communiquées aux concepteurs de l'interface de supervision et du programme de commande qui sont en charge de les implémenter et de les intégrer au système de contrôle-commande.

Figure 3 Exemple de navires comportant le système EdS

La Figure 4 présente l'organisation des chapitres du présent mémoire composé de cinq chapitres. Après avoir présenté au chapitre 1 un état de l'art sur les méthodes de conception et les formalismes de spécifications nous décrivons les problématiques auxquelles nous devons faire face.

Le chapitre 2 présente un état de l'art sur les méthodes de conception de modèles de tâches complexes pour aboutir à une proposition permettant de faciliter l'obtention de ces modèles.

[3] EdS : Eau douce Sanitaire

Le chapitre 3 quant à lui, présente un état de l'art sur les méthodes permettant de faciliter l'obtention des spécifications fonctionnelles des systèmes de contrôle-commande complexes et la génération d'interfaces.

Le chapitre 4 présente l'implémentation d'un flot de conception permettant la mise en œuvre de nos propositions de spécification.

Le chapitre 5 illustre notre démarche sur un cas d'application concret et présente les validations expérimentales réalisées.

Pour terminer, une conclusion rappelle les contributions des travaux de recherches présentés dans ce mémoire et ouvre vers les perspectives qui en découlent.

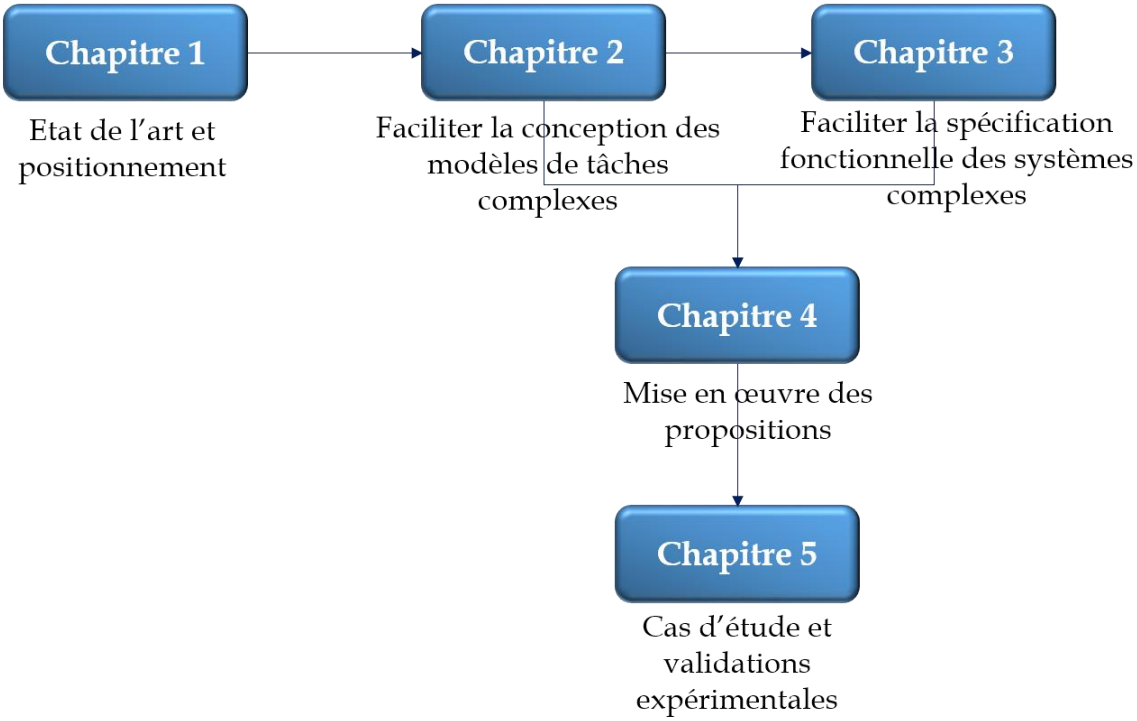


Figure 4 Organisation du mémoire

Chapitre1 : Généralités sur la conception des systèmes de contrôle/commande

Depuis les années 1960, avec l'évolution des infrastructures, les systèmes de contrôle-commande ont envahi le monde industriel. Ces systèmes se doivent d'être modulables à l'avancée des innovations technologiques. Ils sont dotés d'une Interface Homme-Machine (IHM) de supervision destinée au pilotage de procédés physiques (systèmes de gestion de fluide, système de chauffage par géothermie, etc.) et de codes de commande exécutés par des automates. Ils affichent l'état du procédé extérieur, reçoivent des consignes de l'opérateur à travers l'IHM, traitent ces données et, envoient des ordres aux actionneurs du procédé extérieur tout en respectant les contraintes de temps. Sur une IHM de supervision, l'opérateur peut réaliser deux types de commandes : les *commandes de bas-niveau* qui conduisent à des interactions, élément par élément, sur le système (par exemple, l'ouverture ou la fermeture d'un élément du système) et les *commandes de haut-niveau* qui permettent la gestion des fonctions du système (par exemple, la fonction qui permet de transférer du fluide d'une soute à l'autre pour un système de gestion de fluide).

La conception générale d'un système de contrôle-commande et de ses équipements répond à des spécifications imposées par le procédé, par les exigences de sûreté et par les conditions d'exploitation**.** Pour répondre aux besoins de plus en plus accrus dans les secteurs de l'industrie, le développement de ces systèmes doit être rapide, fiable et efficace. Dans la littérature il existe plusieurs approches qui sont utilisées pour concevoir ces types de systèmes, ce qui fait l'objet de ce chapitre.

La *première section* de ce chapitre décrit les approches de conception existantes. Nous nous intéressons aux **approches classiques** les plus connues de l'industrie ; puis aux **approches enrichies sous l'angle des IHM** car un système de contrôle-commande donne une place prépondérante à l'humain, dans un contexte de sûreté de fonctionnement. En effet, la sûreté de fonctionnement est une notion générique qui se rapporte à la qualité de service délivré par un système, de manière à ce que l'utilisateur ait en lui une confiance justifiée.

Sans prise en compte de l'humain, L'**approche Ingénierie Dirigée par les Modèle (IDM)** que nous introduisons enfin**,** permet de construire des systèmes de qualité, par l'utilisation des modèles. Cependant, la prise en compte de l'humain dans cette approche, rend la production de ces systèmes beaucoup plus dure.

Pour tenter d'apporter une solution à ce problème, nous nous intéressons donc, dans *la deuxième section* de ce chapitre, aux différents **formalismes de spécifications** existants. En effet, l'analyse des besoins et la définition des spécifications sont une phase importante présente dans chacune des approches de conception. Deux types de spécifications sont identifiés pour la conception des systèmes de contrôle-commande : **spécification fonctionnelle du système** et **la modélisation des activités de l'utilisateur sur le système**. Ces deux types de spécifications sont distincts mais complémentaires. Dans les deux cas, différents formalismes sont étudiés afin de mettre en évidence dans *la troisième section*, les **verrous** traités dans ce mémoire.

1 La conception des systèmes de contrôle-commande

Les systèmes de contrôle-commande sont des systèmes interactifs de type SCADA (Supervisory Control and Data acquisition), qui permettent de superviser et de contrôler des processus et infrastructures industriels. La conception de tels systèmes suit une démarche classique qui nécessite d'importantes ressources humaines et logicielles. Cette démarche de conception peut être décomposée en quatre grandes étapes dont la première concerne l'analyse de la plateforme à concevoir pour recueillir des informations sur les capteurs, les actionneurs, les algorithmes de commande, etc. qui seront utilisées par le système de contrôle-commande. Au cours de la deuxième étape, des experts métiers spécialisés construisent le schéma mécanique avec des logiciels de conception assistée par ordinateur. Cette conception se fait suivant les normes de l'industrie pour éviter tout problème lors de la phase d'installation du système. La troisième étape consiste généralement à développer les codes de contrôle-commande qui seront exécutés sur les automates programmables. Le développement de l'IHM, l'interface entre les automates programmables et l'utilisateur, se réalise au cours de la dernière étape. Ces différentes étapes sont détaillées dans (Bâra, Popescu, et Lupu 2012). Pour contrôler l'enchainement des activités liées à la conception d'un système de contrôle-commande, les industriels se basent sur différentes méthodes existantes.

1.1 Les méthodes de conception issues du Génie Logiciel

Apparu dans les années 1970, le génie logiciel (GL) (Jaulent 1994) a été mis en place par (OTAN 1968) pour favoriser la conception de composants logiciels de qualité tout en respectant les spécifications du cahier des charges, le coût et les délais. Les facteurs définis pour la qualité d'un logiciel conduisent les industriels à se tourner vers le GL pour la conception de leurs applications.

L'ensemble des activités conduisant à la conception d'un produit est appelé *Cycle de vie* de ce dernier. Les différentes phases du cycle de vie d'un logiciel sont essentiellement : la phase d'analyse du besoin, la phase de spécification, la phase de conception, la phase d'implémentation, la phase de test et la phase de maintenance. Ces phases sont structurées différemment suivant le modèle de conception choisi.

1.1.1 Le modèle en cascade

Le modèle en cascade (Boehm et al. 1988) formalisé dans les années 70, définit des phases séquentielles dans le cycle de vie du logiciel. À l'issue de chaque phase, des documents sont produits pour vérifier et valider la conformité avant de passer à la phase suivante. Sur la Figure 5, le processus débute par des itérations successives entre l'étape de faisabilité et d'analyse des besoins. À l'issue de cette analyse, le cahier des charges contenant toutes les spécifications du système est rédigé. Ces spécifications serviront de référence pour l'étape suivante de conception. Le point d'entrée pour le début d'une phase est le résultat de la phase précédente. Les flèches ascendantes permettent d'exprimer le fait qu'une étape ne remet en cause que l'étape précédente.

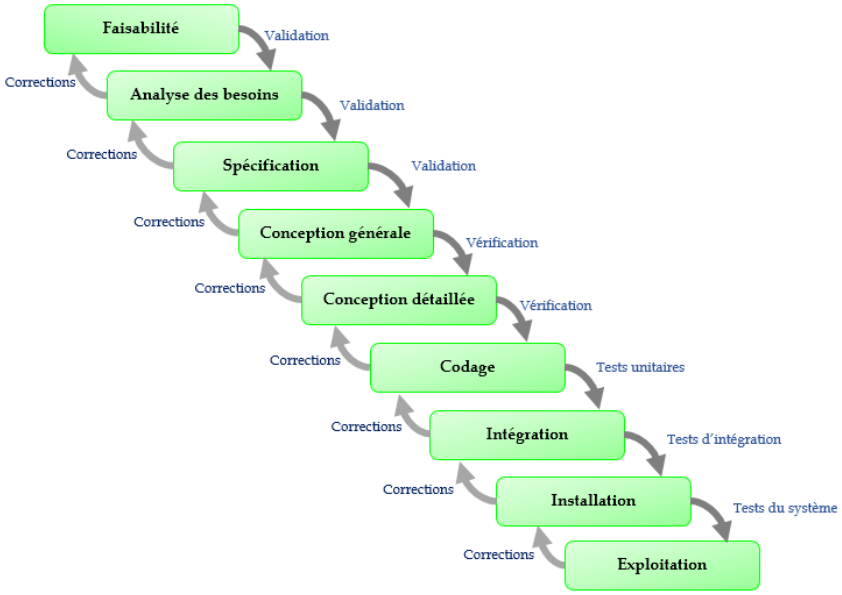


Figure 5 Modèle en cascade

La version plus récente du modèle en cascade offre la possibilité de retour en arrière. Malgré cette modification apportée sur le modèle en cascade, il n'est adapté à la conception que si les spécifications sont précises dès le début du projet et sont susceptibles de ne subir que très peu de changement. Des spécifications stables sont rares dans la conception de systèmes. De plus, le test du système est réalisé très tardivement, à la fin du projet. Les efforts de corrections sont considérables et il est généralement trop tard pour faire des modifications si le système ne répond pas aux besoins de l'utilisateur. Le modèle en cascade est donc souvent abandonné au profit du modèle en V.

1.1.2 Le modèle en V

Le modèle en V (McDermid et Ripken 1984) vient combler les lacunes du modèle en cascade. Dans ce modèle (Figure 6), les phases du processus de développement ne sont plus séquentielles car il existe une cohérence entre les éléments de même niveau qui permet de s'assurer que le système progresse vers un produit répondant aux besoins initiaux. Il permet également de limiter les retours aux étapes précédentes.

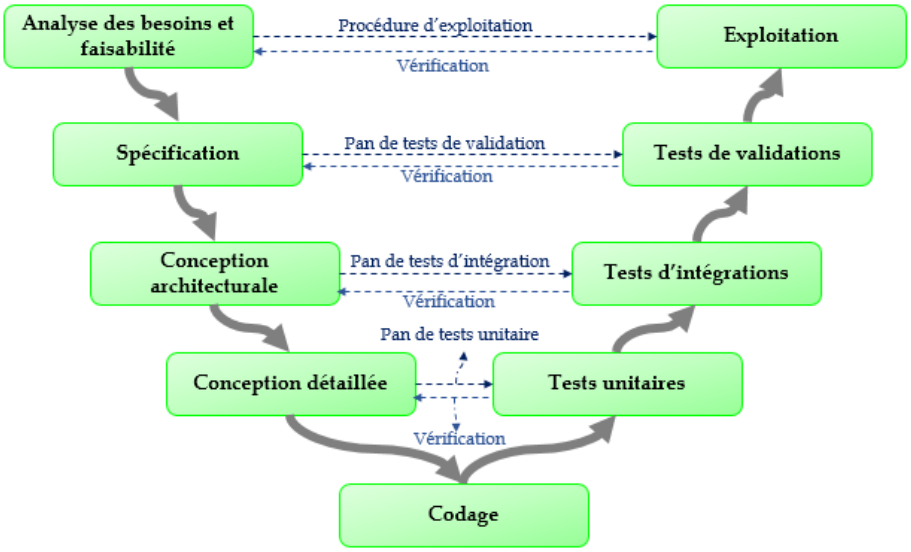


Figure 6 Modèle en V

Ce modèle permet d'anticiper, dans les étapes descendantes, les objectifs des étapes ascendantes de même niveau. Par exemple, les objectifs des tests de validation sont définis lors des spécifications et les objectifs des tests unitaires sont définis lors de la conception. Aussi, les étapes montantes renvoient des informations sur les étapes de même niveau (en vis-à-vis) en cas de défaut pour les corriger. Cependant, il est parfois difficile d'appliquer rigoureusement ce modèle surtout lorsqu'il y a d'important changements dans les spécifications, et ce, dans une phase avancée du projet.

1.1.3 Le modèle en spirale

La particularité du modèle en spirale (Figure 7) proposé par (Boehm et al. 1988) est l'analyse des risques au début de chaque boucle représentant chaque étape du cycle en V. Le processus de traitement de chaque boucle de spirale se fait en quatre étapes : détermination des objectifs, identification des risques et leur réduction, développement et vérification/validation (par prototypage, simulation...), puis planification de la boucle suivante. Boehm a défini quelques risques principaux avec les solutions correspondantes.

Avec les méthodes utilisées, ce modèle permet de sortir un produit de plus en plus robuste. Toutefois, il est difficilement applicable dans une relation client/fournisseur ordinaire. En effet, avant le début de chaque projet un accord est signé entre le client et le fournisseur. Cet accord doit tenir compte de tous les risques encourus pour déterminer les coûts du produit sauf dans le cas où des contrats partiels sont signés à chaque boucle par les deux parties. Dans la plupart des cas, ce modèle ne s'applique que lorsque le client et le fournisseur font partie de la même entreprise (Hugues 2002).

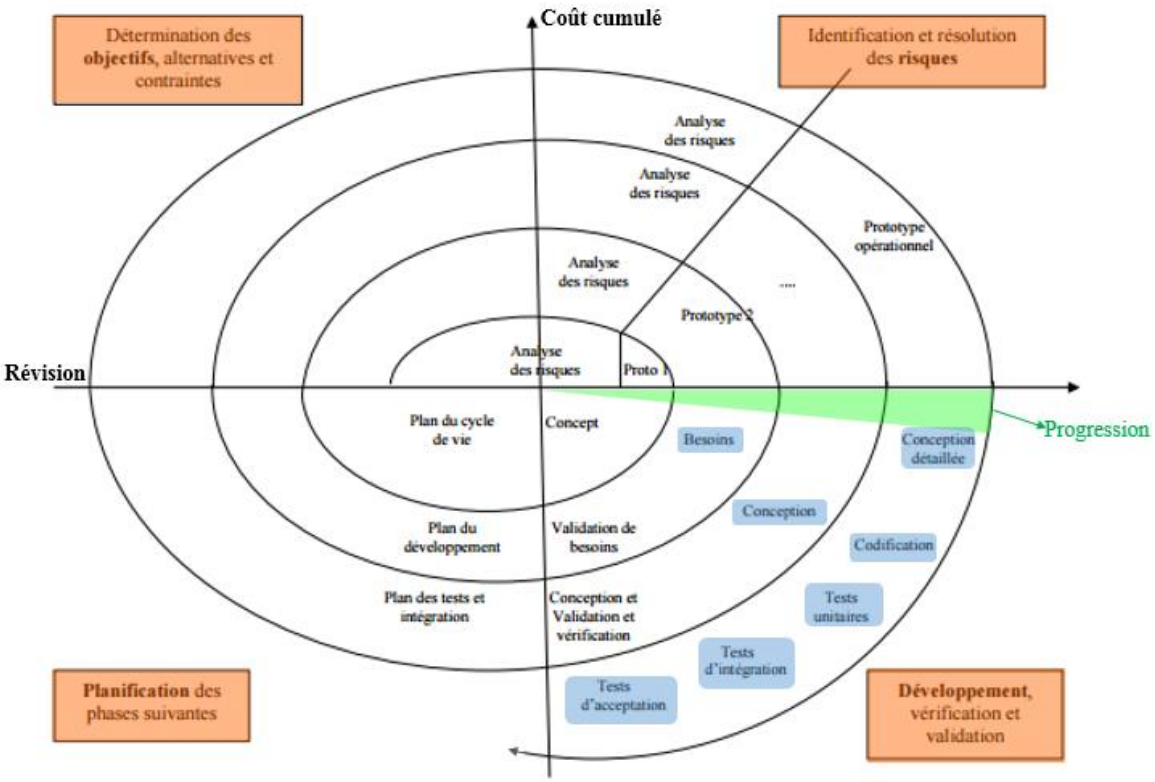


Figure 7 Modèle en spirale

1.1.4 Les méthodes agiles

Les méthodes agiles[4] sont proposées pour faciliter la prise en compte des modifications de la conception initiale. Le terme « agile » fait référence à la capacité d'adaptation aux changements de contextes et aux modifications de spécifications intervenant pendant le processus de développement. Les méthodes agiles ne sont adaptées qu'à des projets limités dans le temps. Elles ne sont pas faites pour des systèmes destinés à être maintenus, à être utilisés par plusieurs clients avec des besoins différents. De plus, la valorisation de la communication directe, par les petites équipes, dans un environnement changeant favorise l'utilisation des méthodes agiles. Ces méthodes ne sont donc pas adaptées à l'ensemble des contextes possibles (Khalil 2011).

1.1.5 Synthèse sur les méthodes classiques de conception

Les cycles de vie classiques du génie logiciel tels que le modèle en cascade, le modèle en V, le modèle en spirale, etc. offrent des approches traditionnelles de conception qui sont divisées en diverses phases. Ces approches permettent de construire des systèmes plus fiables, moins coûteux tout en respectant les délais. Toutefois, en dépit de leurs avantages, ces modèles présentent quelques inconvénients. Pour les modèles en cascade et en V, il n'est pas possible d'identifier facilement les erreurs dans la phase de définition des besoins. De plus, après la phase de spécification, on ne retrouve l'utilisateur qu'à la livraison du système. Ces deux modèles ne sont donc pas adaptés aux projets où les phases de spécification et de conception sont souvent remises en cause.

Pour résoudre ce problème, le modèle en spirale propose une méthode permettant la limitation des risques et des prototypages des versions successives du système. La modification des spécifications conduit à une nouvelle itération dans le modèle. Le nombre d'itérations dépend fortement du système à construire. Ce modèle peut devenir lourd en fonction des modifications de spécifications, ce qui peut conduire à des projets coûteux.

Pour la construction d'un système, il n'existe pas de modèle idéal, tout dépend des caractéristiques opérationnelles de ce dernier. Les systèmes de contrôle-commande de procédés diffèrent fondamentalement des systèmes informatiques classiques. Leur conception nécessite la mise en œuvre de méthodes et de techniques appropriées. Les phases présentées dans les différents modèles du génie logiciel ne reflètent pas réellement les activités menées au cours d'un projet de conception de système de contrôle-commande (Bignon 2012). La réalisation de ce type de système conduit à la conception simultanée de codes pour automates programmables et d'IHM. Les approches traditionnelles ne tiennent pas compte des étapes essentielles dans la conception des systèmes interactifs (BARTHET 1988). Avec les modèles classiques issus du génie logiciel, les fonctionnalités des systèmes sociotechniques complexes sont priorisés au détriment des utilisateurs. Avec les approches classiques de conception de systèmes de contrôle-commande, l'interface et les interactions ne sont définies qu'à la fin du cycle. Il est indispensable lors de la conception des systèmes interactifs de prendre en compte l'utilisateur dès le début de la conception afin de faciliter l'utilisabilité et l'acceptabilité du système. Pour ce faire, les modèles enrichis sous l'angle des IHM intègrent des notions inexistantes (ou peu visibles) dans les modèles classiques, qui concernent les étapes

[4] http://agilemanifesto.org/iso/fr/manifesto.html

importantes d'analyse et de modélisation des tâches utilisateurs (Kolski, Ezzedine, et Abed 2001).

1.2 Les modèles de conception enrichis sous l'angle des IHM

L'enrichissement des approches classiques s'est avéré nécessaire pour intégrer l'utilisateur dans le processus de conception du système. Parmi les modèles enrichis sous l'ange des IHM présentés dans (Kolski, Ezzedine, et Abed 2001), nous allons nous intéresser au modèle en étoile, au modèle de Long, au modèle en U et au modèle Nabla. Ces modèles sont mieux adaptés à la conception des systèmes interactifs, que les méthodes classiques.

1.2.1 Le modèle en étoile

Le modèle en étoile (Figure 8) (Hartson et Hix 1989) place l'évaluation au centre de la conception faisant intervenir ainsi l'utilisateur à toutes les étapes du cycle afin de détecter au plus tôt les problèmes liés à l'utilisabilité. Le cycle de vie de ce modèle propose de réaliser les différentes phases par itération avec l'évaluation, sans ordre imposé. Cependant, dans la pratique, l'implémentation est réalisée à la fin du cycle.

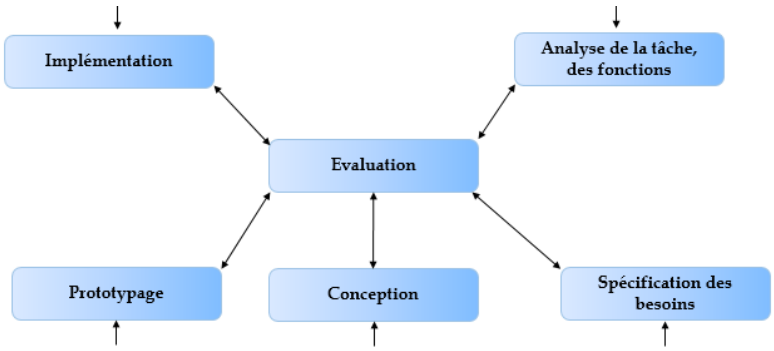


Figure 8 Modèle en étoile

Bien que ce modèle offre la conception centrée utilisateur recherchée, le processus de développement du cycle de vie est lent et l'équipe de conception doit être maintenue tout au long de ce processus.

1.2.2 Le modèle de Long

Long s'appuie sur le principe de conception et de réalisation des IHM pour proposer un modèle proche du modèle en cascade. Dans la pratique, on retrouve le cycle usuel « spécification – mise en œuvre – évaluation ». Compte tenu des changements qui peuvent intervenir dans les spécifications pendant le cycle de vie du projet, une évaluation avec retour en arrière est exigée par le modèle de Long (Long et Denley 1990). Sur la Figure 9, les rectangles représentent les étapes de développement et les ovales sont les documents produits. Bien que ce modèle soit proche du modèle en cascade, la conception de l'IHM est précisée et l'évaluation permet de prendre en compte assez tôt les besoins de l'utilisateur.

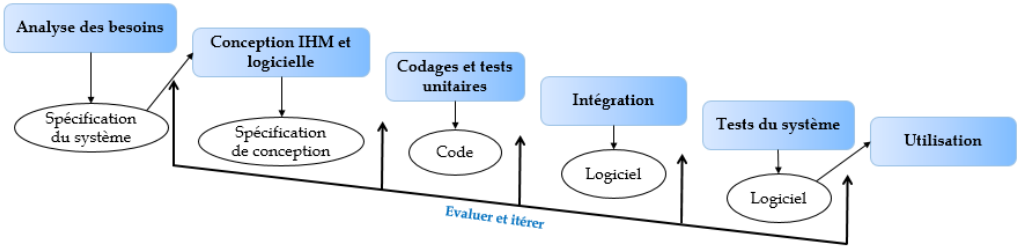


Figure 9 Modèle de Long

1.2.3 Le modèle en U

Dans le modèle en U (Figure 10) (Abed 1990), en plus de l'évaluation, les facteurs humains sont pris en compte. En effet, l'analyse et la modélisation de la tâche et de l'utilisateur y sont précisées. Les étapes de conception sont différentes de celles des modèles classiques issus du génie civil. Le modèle en U est structuré en deux phases formant le « U » : une phase descendante portant sur l'analyse, la spécification et la conception de l'IHM ; et une phase ascendante axée sur l'évaluation du système global en s'appuyant sur des critères d'efficacité et des critères centrés sur l'utilisateur. Une étape importante de Validation/Affinage consiste à confronter le modèle de tâches théorique de la phase descendante à celui des activités réelles de la phase ascendante. À partir des résultats de cette opération, l'IHM est affinée ou validée.

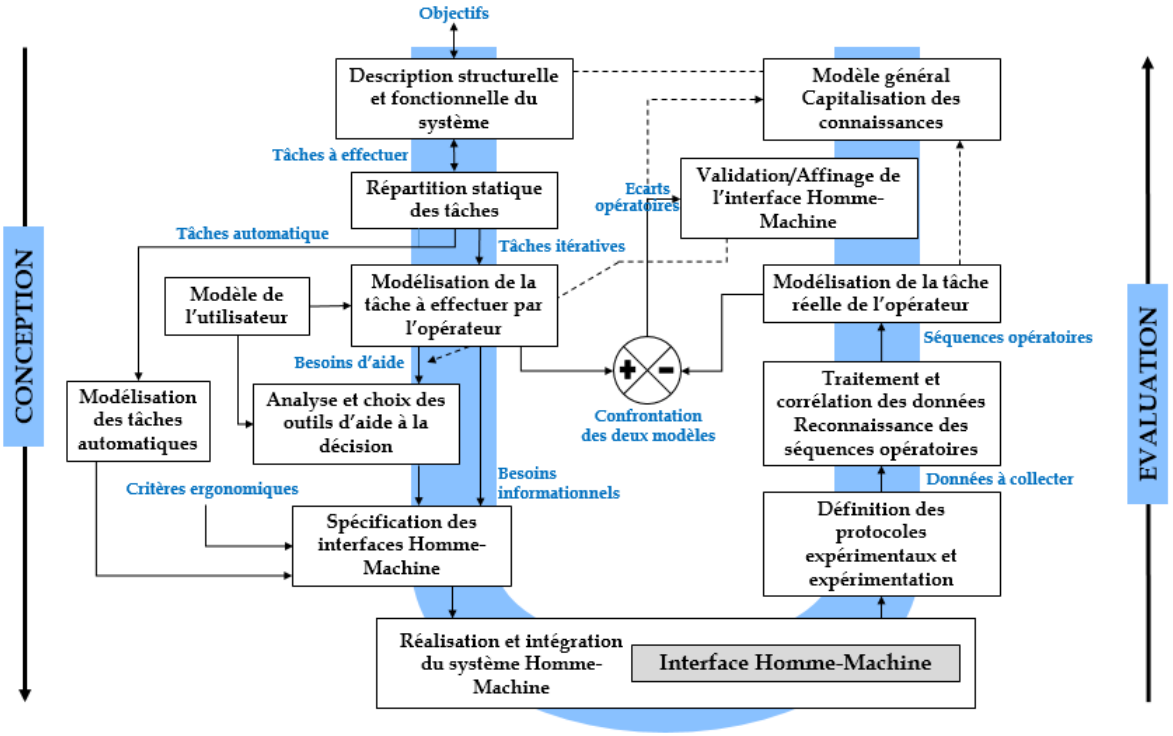


Figure 10 Modèle en U

Ce modèle a prouvé son application à plusieurs domaines complexes tels que le contrôle aérien, les applications chimiques et ferroviaires (Lajnef, Ayed, et Kolski 2005).

1.2.4 Le modèle Nabla

Comme le modèle en U, le modèle Nabla (Figure 11) (Kolski 1997) tient également compte des facteurs humains dans le cycle de vie du projet. La particularité du modèle Nabla est qu'il sépare les étapes de conception de l'interface (la partie gauche du modèle) de celles de conception des modules applicatifs ou d'aide (partie droite du modèle). Les deux parties du modèle suivent les mêmes étapes que celles décrites en génie logiciel. En réalité, Nabla représente un double cycle en V qui repose sur la confrontation progressive entre un modèle réel et un modèle de référence (système interactif idéal). Cette confrontation permet d'obtenir des données pertinentes pour la spécification du système interactif. Les spécifications obtenues sont ensuite évaluées et validées. En effet, des phases d'évaluation et de validation s'appuyant sur des spécifications ergonomiques et socio-ergonomiques se situent au niveau de chaque étape des deux modules afin de garantir l'utilisabilité du système interactif à concevoir.

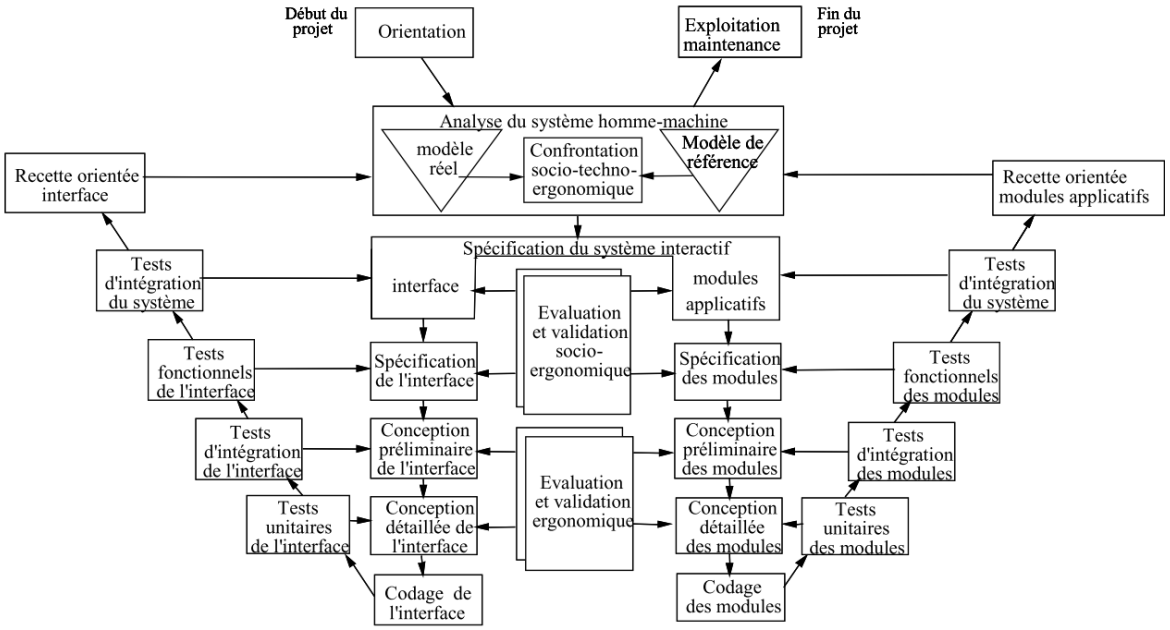


Figure 11 Modèle Nabla

1.2.5 Synthèse sur les méthodes enrichies sous l'angle des IHM

Pour répondre aux besoins de conception dans le domaine des IHM, les modèles classiques ont été étendus, adaptés ou revus pour obtenir les modèles enrichis sous l'angle des IHM. Quatre de ces modèles sont présentés ici : le modèle en étoile, le modèle de Long, le modèle en U et le modèle Nabla. Le modèle en étoile est un modèle très flexible qui n'impose pas d'ordre pour la réalisation des étapes du cycle de vie du projet. Le modèle de Long est proche d'un modèle classique mais ne précise pas les activités liées à la modélisation de la tâche et de l'utilisateur. Dans le modèle en U, le développement itératif se fait tâche par tâche à partir de la confrontation de la tâche prescrite à la tâche réelle, et en fonction des résultats de cette confrontation. Dans le modèle Nabla, la modélisation de l'utilisateur, la modélisation des tâches humaines et les relations entre ces deux modèles et les spécifications ne sont pas clairement indiquées.

Les modèles enrichis sous l'angle des IHM ne fournissent pas forcément l'assurance qu'un projet de conception de système interactif soit réalisé avec un succès total (Ltifi 2011), mais ils tiennent compte des facteurs humains qui sont essentiels dans la conception des systèmes interactifs. L'inconvénient principal de ces modèles réside, d'une part, d'un point de vue méthodologique, dans les aspects fondamentaux tels que la modélisation des tâches humaines, le développement itératif des prototypes, et l'évaluation du système interactif (Kolski, Ezzedine, et Abed 2001). En effet, la prise en compte de ces aspects est essentielle dans le développement d'un système interactif. D'autre part, face à la complexité des systèmes interactifs, plus particulièrement les systèmes de contrôle-commande, où le nombre d'intervenants et la diversité des domaines d'études sont considérables, l'utilisation de ces modèles entraine des coûts élevés de développement car des problèmes de cohérence d'ensemble et d'interprétation des spécifications conduisent à des erreurs. Bien que ces modèles proposent dans le cycle de vie de conception du système, des phases d'évaluation pour la détection et la correction des erreurs, ces phases peuvent nécessiter d'importants efforts de re-conception. En effet, il est très rare d'avoir un système correctement réalisé du premier coup. De ces constats découlent les approches basées sur les modèles permettant aux

concepteurs d'exprimer leurs expertises par des modèles, puis d'utiliser des méthodes appliquant les techniques de l'Ingénierie Dirigée par les Modèles (IDM) pour réaliser des passerelles entre les modèles et automatiser le processus de conception.

1.3 L'IDM pour la conception des systèmes de contrôle-commande

Dès le début du XXIème siècle, l'ingénierie logicielle s'est résolument orientée vers l'IDM qui permet aujourd'hui d'exploiter les modèles de conception comme une matière de production grâce à des techniques de génération automatique. Pour ce faire, l'IDM introduit les notions de *modèles*, et de *méta-modèles* définis sur 4 niveaux, en partant du monde réel jusqu'au méta-métamodèle (Figure 12).

En 1968, Marvin Minsky donne la définition suivante : « *Un modèle est une abstraction, une simplification d'un système qui est suffisante pour comprendre le système modélisé et répondre aux questions que l'on se pose sur lui.*» (Minsky 1968). Un système peut-être décrit par différents modèles liés les uns aux autres (Figure 12). À travers cette définition, on voit apparaître la relation de *représentation* qui stipule qu'un modèle représente un système du monde réel. Un *méta-modèle* représente la syntaxe utilisée pour décrire les modèles. La syntaxe d'un méta-modèle fournit les éléments nécessaires pour décrire un modèle qui lui est conforme. La relation de *conformité* exprime le fait qu'un modèle doit être conforme à son méta-modèle et aussi qu'un méta-modèle est conforme à un méta-métamodèle (Figure 12).

Le principe de l'IDM repose sur la génération de tout ou partie d'une application par utilisation de transformations de modèles. Plusieurs transformations peuvent être mises en œuvre jusqu'à l'obtention des artéfacts exécutables. Chaque *transformation* permet de générer des modèles cibles à partir de modèles sources. Il existe deux types de transformations : les transformations endogènes et les transformations exogènes. Une transformation est dite **endogène** lorsque les modèles sources et cibles sont conformes à un même méta-modèle. Par contre, une transformation est dite **exogène** lorsque les modèles cibles sont conformes à un méta-modèle différent de celui des modèles sources.

L'utilisation des modèles et des transformations de modèles offre un développement souple à différents niveaux du génie logiciel, dans un cadre méthodologique unique qu'est l'IDM. L'IDM joue donc un rôle d'unificateur vis-à-vis des différentes activités du génie logiciel. En effet, les cycles de vie du génie logiciel présentés dans la section 1.1 de ce chapitre englobent plusieurs phase, qui produisent chacune un ou plusieurs artéfacts (codes sources, fichiers de configurations etc…). Un des problèmes du génie logiciel est d'assurer la cohérence et une traçabilité entre ces différents artéfacts(Jézéquel, Gérard, et Baudry 2006).

Alors que l'IDM s'applique en premier lieu sur les problématiques du génie logiciel, l'IHM se tourne au cours des années 1990 tout d'abord vers les systèmes à base de modèles (Model based-system) pour générer de façon semi-automatique les IHM. En raison principalement de la qualité des IHM produites ,l'approche n'a pas connu le succès espéré (Sottet et al. 2006). Le domaine des IHM s'est ensuite tourné vers le développement à base de modèles (Model-based User Interfaces (MBUI) (Gerrir Meixner, Paternò, et Vanderdonckt 2011) pour résoudre les problèmes de conception rencontrés. Le MBUI offre la possibilité aux concepteurs de se concentrer sur les aspects les plus important de la conception en identifiant les modèles de haut-niveau permettant d'analyser et de spécifier les systèmes interactifs (Gerrit Meixner, Calvary, et Coutaz 2013). Ces modèles sont ensuite utilisés dans des outils intégrant les techniques de MBUI pour générer les interfaces utilisateurs.

Les recherches dans le domaine de l'IHM se basent de plus en plus sur un ensemble de modèles et de transformations pour la conception des IHM. Il est aujourd'hui admis que les concepts de l'IDM s'appliquent parfaitement à ce domaine. En effet, l'application du paradigme de modèle au développement des IHM permet de réduire le coût de développement tout en restant indépendant des langages d'implémentation, d'obtenir des IHM de meilleure qualité, de détecter plus rapidement les erreurs de conception et d'intégrer les connaissances de l'expert métier dans les modèles exécutables (Sottet et al. 2006).

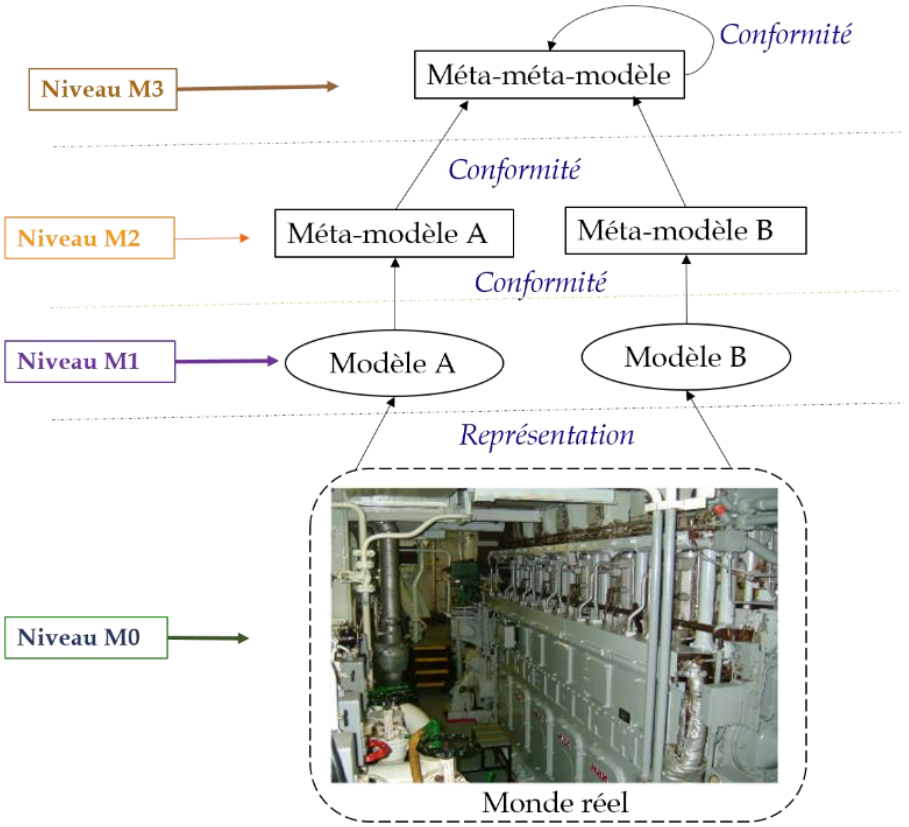


Figure 12 L'approche de l'IDM

Nous nous intéressons, tout particulièrement, aux travaux portant sur les interfaces de contrôle-commande. Nous nous intéressons ensuite aux approches permettant de générer les codes de commandes. Enfin, nous nous intéressons aux travaux de génération conjointe de codes de commande et d'IHM de supervision dans lesquels s'inscrit cette thèse.

1.3.1 Méthodes et outils de génération d'IHM

L'utilisation de l'IDM dans le domaine de l'IHM est un sujet en plein essor, dont l'historique et l'évolution sont présentés dans (Gerrir Meixner, Paternò, et Vanderdonckt 2011). CAMELEON[5] (Calvary et al. 2003) est ainsi devenu un modèle de référence d'architecture pour la production d'IHM plastiques en utilisant les techniques de MBUID. Il recommande quatre modèles fondamentaux (Figure 13) : le modèle de tâches et de concepts métiers, l'interface abstraite, l'interface concrète et l'interface finale. Ces termes sont plus détaillés dans (Coutaz et al. 2012).

[5] Context Aware Modelling for Enabling and Leveraging Effective interactiON

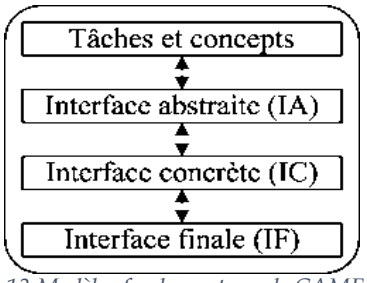


Figure 13 Modèles fondamentaux de CAMELEON

La description des modèles dans le but de générer progressivement de façon semi-automatique les systèmes interactifs a été utilisée dans plusieurs travaux comme recensé dans (Szekely 1996). Compte-tenu de la spécificité des systèmes contrôle-commande, nous avons choisi de ne présenter qu'un seul système généraliste, pour nous concentrer sur les systèmes ayant un rapport certain avec la supervision et le contrôle-commande. Ainsi, nous présentons TRIDENT (Vanderdonckt 1995), un système qui permet de générer de façon automatique ou semi-automatique des interfaces. D'autres telles que ERGO-CONCEPTOR (Moussa, Kolski, et Riahi 2000) sont plus axés sur la génération des interfaces de contrôle-commande.

1.3.1.1 Le projet TRIDENT

La méthode TRIDENT (Tool foR Interface Developement EnvironmeNT) (Vanderdonckt 1995) propose une approche basée sur les modèles pour le développement des systèmes interactifs. Elle offre un environnement de génération automatique des interfaces.

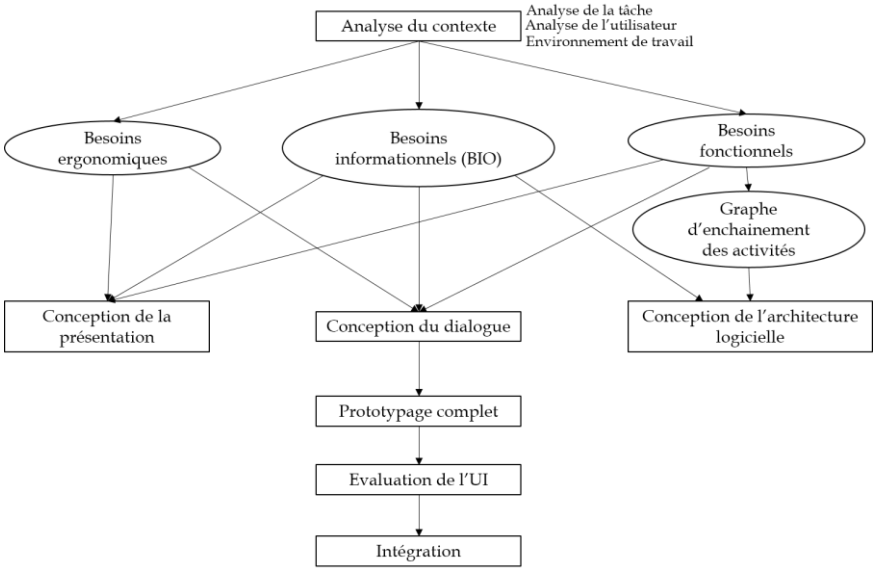


Figure 14 Structure méthodologique de TRIDENT

TRIDENT se base sur l'analyse du contexte qui comprend l'analyse de tâche de l'utilisateur et de l'environnement de travail pour identifier les besoins ergonomiques, informationnels (BIO) et fonctionnels. Le graphe d'enchainement des activités est ensuite déduit des besoins fonctionnels pour être utilisé dans la spécification de la présentation et de l'architecture. L'objectif étant de fournir une structure générale qui englobe différents styles d'interaction et différentes structures de dialogue. Cette structure sera affinée en fonction du contexte. En plus de ne pas être adaptée à la génération de système de contrôle-commande, le passage, dans TRIDENT, de l'analyse de tâche à l'identification des besoins informationnels (BIO), puis à la spécification de la présentation n'est pas précisé. Les méthodes utilisées manquent encore de formalisme stable (MOUSSA 2005).

1.3.1.2 La démarche ERGO-CONCEPTOR

Le développement à base de modèles est présent depuis quelques années dans les projets de conception d'IHM de supervision (Allegre 2012). Afin d'éviter les erreurs humaines sur des tâches de haut niveau telles que la surveillance globale, en particulier dans le domaine du contrôle de procédés industriels, la méthode ERGO CONCEPTOR (Moussa, Kolski, et Riahi 2000) a été proposée. Elle est composée de trois modules (Figure 15) : un module pour la description du procédé, un module pour la génération d'un fichier de spécification et un module pour la génération des vues graphiques. Le premier module sert à décrire le procédé à surveiller en détail. À partir de cette description détaillée du procédé à surveiller, le deuxième module génère les spécifications qui serviront de base à la conception de l'IHM de supervision. Le troisième module exploite donc ces spécifications pour générer des vues graphiques de contrôle de procédé. Lors de la conception de l'IHM, le concepteur peut choisir le mode de représentation selon les vues graphiques à concevoir (Figure 15).

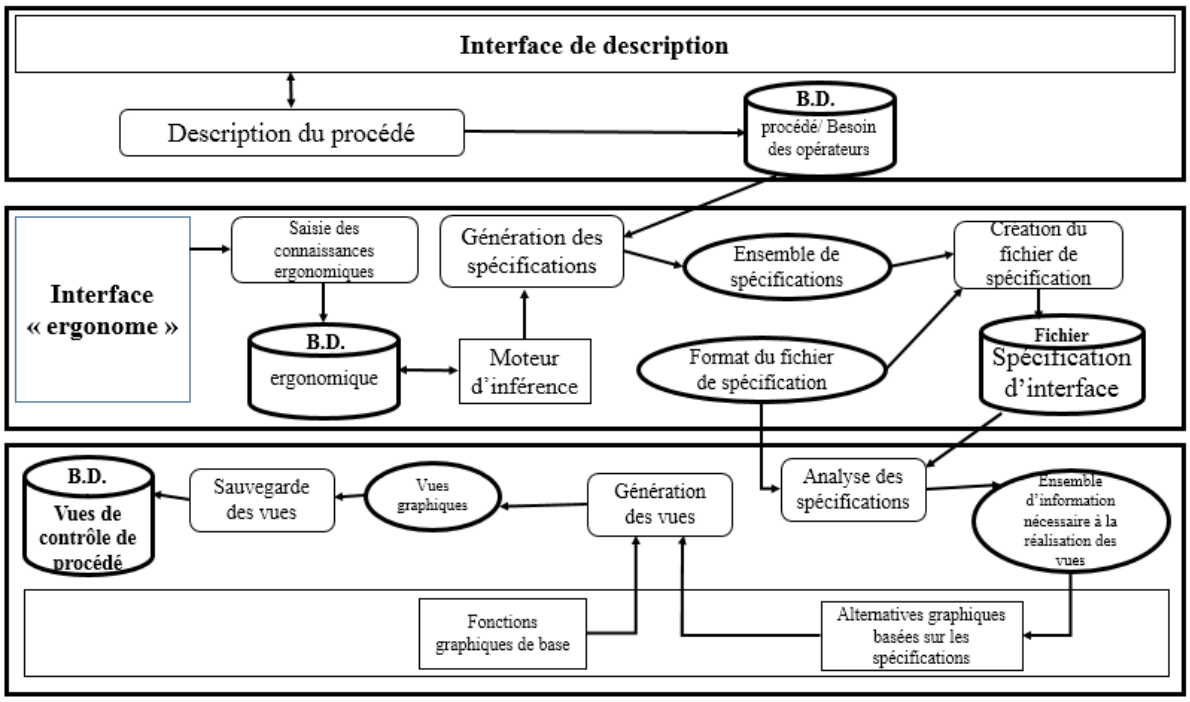


Figure 15 Architecture du système ERGO-CONCEPTOR

Malgré ces nombreux avantages (Kolski 1993) pour le monde industriel, ERGO CONCEPTOR présente certaines limites. D'une part, la première phase de la conception qui est la description du procédé par page-écran textuelle est fastidieuse, source d'interrogation et d'oublis. D'autre part, elle ne permet pas une génération automatique du système. De plus, les vues générées par ERGO-CONCEPTOR sont des vues statiques car elles ne gèrent pas le dialogue homme-machine, ce qui rend l'interface incomplète. Pour prendre en compte la dynamique du système, Ergo-Conceptor+ a été proposé (MOUSSA 2005). Dans ERGO-CONCEPTOR+, la phase fastidieuse de description du procédé est remplacée par une modélisation de procédé utilisant la méthode SADT (D. A. Marca et McGowan 1987) pour décomposer le système en sous-systèmes élémentaires. Ensuite, une analyse de dysfonctionnement est modélisée sur chaque sous-système. Cette analyse de disfonctionnement est suivie d'une analyse de la tâche opérateur, réalisée pour chaque sous-système, que ce soit en fonctionnement normal ou anormal. Enfin, le processus d'interaction homme-machine est modélisé avec des Réseaux de Petri Interprétés (RdPI) permettant d'exprimer le dialogue homme-machine.

Les trois premières phases de cet outil montrent clairement une frontière virtuelle entre la production des données issues des analyses de fonctionnement normal et de dysfonctionnement, et l'exploitation des données produites par les informaticiens pour la spécification du système (MOUSSA 2005). Une mauvaise interprétation et manipulation des données peuvent générer des spécifications imprécises et incomplètes. De plus, la génération du code de commande n'est pas prise en compte.

1.3.2 Génération automatique de codes de commande

En général la conception du code de commande pour les automates programmables est en charge d'un expert automaticien. Il se base sur le cahier de charges (ou sur une analyse fonctionnelle) et sur la représentation du système (schéma mécanique) pour réaliser les codes de commande. Cette tâche n'est pas aisée, et trouver une bonne adéquation entre la solution fonctionnelle et l'architecture matérielle est un réel défi dans la conception des systèmes de contrôle-commande (Bévan 2013). Des techniques d'analyse et de validation (Lallican 2007) sont utilisées pour détecter les erreurs dans les codes conçus. L'effort de re-conception pour corriger ces erreurs peut être considérable, dépendant de la complexité du système. Face à cela, plusieurs travaux recensés dans (Bévan 2013) proposent diverses approches pour générer les codes de commandes.

Certaines de ces approches exploitent les systèmes multi-agents et/ou holoniques (Leitão 2009). D'autres utilisent des modules de contrôle qui modélisent une partie de la commande pouvant être traduite vers une plateforme spécifique pour être exécutée par un système physique. Malgré les nombreux avantages de ces approches, les codes de commandes qu'ils produisent doivent être traduits dans un langage normalisé avant d'être utilisés, dans les systèmes de contrôle-commande embarqués sur un navire.

Dans la littérature, il est proposé de générer, à partir de modèles spécifiés en UML (Panjaitan et Frey 2006), la commande dans la norme IEC-61499 (Vyatkin 2009). Cette norme est une évolution prometteuse mais peu robuste de la norme IEC 61131-3 (Thramboulidis 2012) et donc n'est souvent pas utilisée dans l'industrie.

1.3.3 Génération conjointe d'interfaces et de codes de commande

Parmi les travaux que nous avons présentés jusque-là, certains traitent uniquement de la génération de l'IHM. D'autres se focalisent sur la génération des codes de commande associés à cette IHM. La cohérence entre les codes générés n'est cependant pas assurée. Pour résoudre ce problème, d'autres travaux plus récents proposent de générer simultanément l'IHM de supervision et les codes de commande.

1.3.3.1 L'outil SCADA CAD

Les travaux de (Bâra, Popescu, et Lupu 2012) permettent de générer l'IHM et les codes de commande pour les systèmes de contrôle-commande. Ces travaux ont conduit à l'implémentation de l'outil SCADA CAD composé d'une *interface de saisie* et d'un système expert pour générer *les configurations des systèmes* de type SCADA. Cet outil permet d'accélérer le processus de conception des systèmes interactifs de supervision en réduisant le temps et le nombre d'intervenants dans la conception des systèmes de contrôle-commande.

1.3.3.1.1 L'interface de saisie et le système expert

L'*interface de saisie* permet aux experts de saisir un ensemble de données d'entrée sous forme de modèle du système SCADA à construire et qui sera utilisé par le système expert. Ce modèle

contient tous les capteurs, actionneurs et les séquences de commandes des éléments pour la réalisation des fonctions du système.

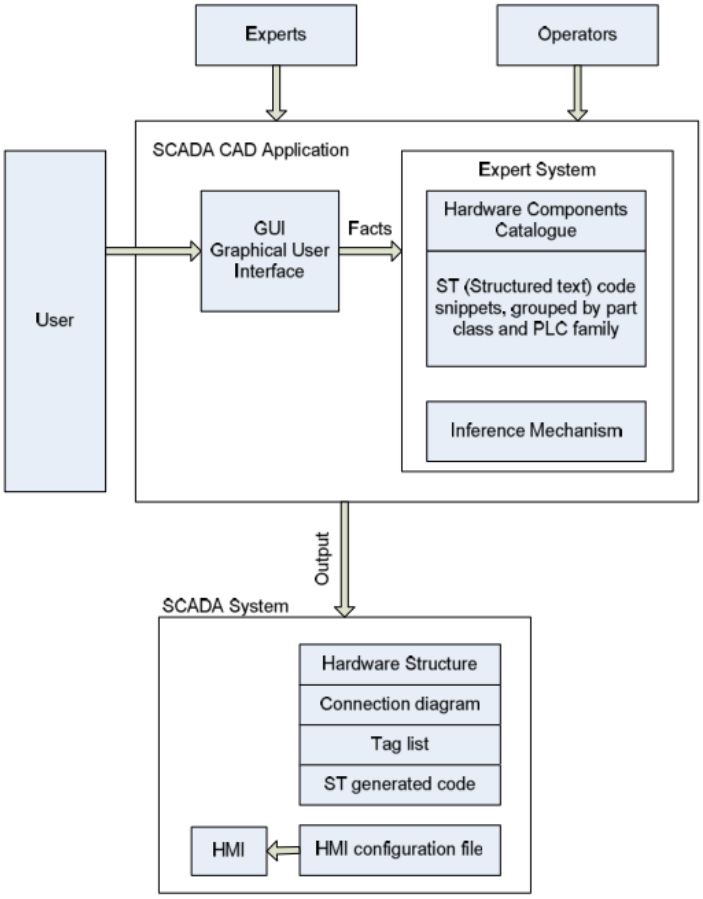


Figure 16 Structure de l'application SCADA CAD

La méthode de saisie de données n'est pas détaillée mais il s'agit de récupérer un ensemble de données qui décrivent comment les divers capteurs et actionneurs sont reliés aux blocs de code qui composent le système. Ces données contiennent les informations relatives à la connexion des différents capteurs et actionneurs aux blocs de code constituant le système de contrôle-commande, les vues graphiques des composants du système définies avec JSON (JavaScript Object Notation), les informations sur le positionnement des composants sur l'IHM, la liste des entrées/sorties etc. Les données recueillies sont structurées dans une base de connaissances exploitée par le système expert. La base de connaissances doit contenir des informations sur les différents composants matériels, les extraits de code en langage ST (Structured Text) pour décrire le comportement des différents types d'éléments, et un ensemble de règles pour gérer la façon dont les informations sont présentées à l'opérateur à travers l'IHM. Un algorithme de génération de système de supervision permet de définir la forme sous laquelle les informations seront structurées. L'algorithme est intégré dans un mécanisme d'inférence qui génère des configurations pour le système.

1.3.3.1.2 Les configurations générées

Les *configurations* contiennent les éléments suivant : la *structure des matériaux du système*, les *diagrammes de connexion*, la *liste de Tag*, le *software* et *le fichier de configuration* de l'IHM. La *structure des matériaux* est une liste complète des composants du système (capteurs, modules d'entrée, interfaces d'entrée et de sortie, automates). Le *diagramme de connexion* est une table

qui décrit toutes les connexions entre les éléments matériels. Chaque connexion doit avoir les caractéristiques suivantes : le point de départ, le point d'arrivée, et le type de câble. La *liste de Tag* est une liste complète des données et variables nécessaires pour configurer correctement l'automate et le serveur OPC qui permet la communication entre l'automate et l'IHM. Le *software* est une collection de programmes qui seront exécutés sur les automates. Le *fichier de configuration de l'IHM* est un fichier qui contient des informations sur la représentation graphique de la plate-forme, une liste des alarmes qui lui sont associées et une liste des entrées et des sorties qui doivent être affichées.

L'outil SCADA CAD met en œuvre des algorithmes à travers un système expert utilisant les techniques d'Intelligence Artificielle pour générer la plupart des tâches liées à la conception des systèmes de type SCADA. Le système expert permet d'interpréter une entité monolithique (un modèle d'entrée) sous une forme reconnue par le système, à partir de laquelle les composantes du système SCADA à générer, peuvent être déduites.

1.3.3.2 *Le projet Anaxagore*

Le projet de recherche *Anaxagore* (Bignon 2012) (Figure 17) est un ensemble de technologies et de formalismes qui propose une solution basée sur l'IDM pour générer automatiquement et conjointement les interfaces de contrôle et les codes de commande composant les systèmes de contrôle-commande. La démarche, centrée sur l'architecture du système à concevoir, s'appuie sur une approche ascendante utilisant des composants sur étagères, intégrant des informations structurées en vues. Des méthodes d'association d'informations et de dérivation de modèles permettent de générer progressivement des modèles d'exploitation (Figure 17). L'objectif de ce projet est de proposer des méthodes et outils de conception innovants qui garantissent la qualité des produits documentaires et logiciels d'un projet, tout en réduisant le temps de conception et de re-conception. Il s'agit d'un environnement d'aide à la conception qui met en œuvre les concepts de l'IDM dans un contexte de supervision industrielle. *Anaxagore* s'est attaché à proposer une passerelle entre les mécaniciens, les informaticiens et les automaticiens.

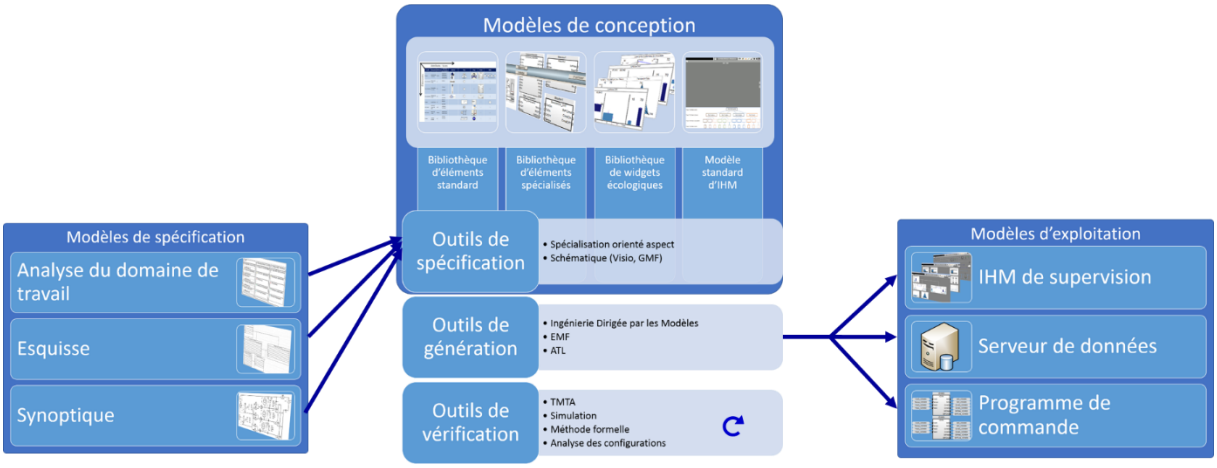


Figure 17 Ecosystème d'Anaxagore

1.3.3.2.1 Les modèles de spécification

Les modèles de spécification sont des modèles d'entrée saisis avec des outils de spécification. Ces modèles peuvent être considérés comme des DSL (Domain Specific Language). Un DSL est caractérisé par un méta-modèle et une syntaxe (Frizon De Lamotte 2006) auxquels des règles supplémentaires peuvent être adjointes. Ainsi, la syntaxe du modèle synoptique est fournie par la norme ANSI/ISA 5.1 (Ansi/Isa 1992), décrite par les experts métier et utilisée

au plus tôt dans tous les projets de conception de système de contrôle-commande. Ce modèle est complété par deux autres modèles (modèle d'analyse du domaine de travail et esquisse) (Rechard 2015) qui sont inspirés par l'approche écologique de conception des interfaces (Vicente 1999).

1.3.3.2.2 Les modèles de conception

Les modèles de conception sont un ensemble des briques mises à disposition des concepteurs dans un souci de standardisation de leur production. Ce groupe de modèles est hétérogène et fait appel à plusieurs paradigmes. La **bibliothèque d'éléments standards** regroupe les unités constitutives du procédé du système (les éléments). Le concept d'élément s'inspire bien évidement du paradigme objet. Mais il est également plus vaste car chaque élément se compose de vues (Lallican 2007) représentant chacune une facette de la conception. Par exemple, la vue de supervision d'un élément correspond au symbole graphique utilisé pour le représenter à l'écran, ainsi que le programme permettant d'animer ce symbole. La bibliothèque permet ainsi de stocker toutes les informations nécessaires à la spécification d'un système, au moins d'un point de vue architectural.

Partant du principe que tous les composants d'un système ne peuvent pas être standardisés, un mécanisme inspiré par la programmation orientée aspect (Kiczales et al. 1997) est proposé. Ce mécanisme permet, à partir d'un élément adaptable, de concevoir un élément spécifique à un projet et de le stocker dans une **bibliothèque d'éléments spécifiques** dont la structure est identique à la bibliothèque d'éléments standards.

La **bibliothèque de widgets écologiques** regroupe des composants graphiques à haut niveau d'abstraction nécessaires à la génération d'une interface écologique. (Liu, Nakata, et Furuta 2002) ont défini 8 processus qui, par composition, permettent de spécifier l'ensemble des domaines de travail causaux. Chaque widget de la bibliothèque renvoie à un processus au sens de l'analyse du domaine de travail.

Le **modèle standard d'IHM** regroupe les règles de présentation de l'interface à produire, il pose ainsi l'identité visuelle du produit fini. En ce sens, c'est un guide de style au même titre que ceux proposés par Microsoft® (Microsoft 2014) ou Apple® (Apple 2013). C'est également un patron de conception puisqu'il impose des règles d'agencement des zones de l'IHM (Conte et al. 2001).

Pour contrôler et valider la qualité des modèles, plusieurs outils de vérification ont été mis en œuvre. Ces outils exploitent les techniques de la TMTA (Turing Machine Task Analysis) (Rechard et al. 2015), les méthodes de vérification formelles (Mesli-kesraoui, Kesraoui, et al. 2016; Mesli-kesraoui, Toguyeni, et al. 2016), les méthodes de vérification par simulation (Prat et al. 2016) et les concepts d'analyse des configurations (Bignon 2012).

1.3.3.2.3 Les modèles d'exploitation

Les outils de génération s'appuient sur le concept de transformation de modèle dont une partie est présentée dans (Mens, Czarnecki, et Gorp 2006; To 2008) pour générer les modèles d'exploitation. Les **modèles d'exploitation** générés s'inspirent du concept de PSM (Plateform Specific Model) proposé dans la recommandation MDA (Model Driven Architecture) de l'OMG (OMG 2003). Il s'agit en réalité d'IHM, de code de commande et des données d'un serveur qui assurent la communication entre la supervision et la commande. Ces programmes permettent à un opérateur de superviser le système décrit par les modèles de spécification (Goubali et al. 2014).

Dans le projet Anaxagore, la conception des différents modules pour la supervision et la commande du système se font par raffinages successifs de différents modèles correspondant aux différentes étapes du processus. La solution s'appuie sur les particularités des systèmes tiers utilisés, tant pour le contrôle du système physique que pour la réalisation de l'application interactive. Elle consiste globalement à construire l'interface de supervision en se basant sur un modèle prédéfini d'interface, puis à déterminer les éléments dynamiques.

1.3.4 Synthèse sur l'IDM

L'utilisation du MBUID et de l'IDM en général, permet de réduire l'écart entre les exigences et l'implémentation, de coordonner le travail des divers intervenants d'un projet, d'améliorer la communication entre les intervenants à travers des modèles explicites, de capturer et réutiliser les connaissances de l'expert tout au long du développement du système afin de réduire les erreurs de conception. De plus, les modèles, méta-modèles et transformations sont réutilisables dans d'autres projets et peuvent être plus facilement maintenus. Toutes ces avantages sont détaillées dans (Hutchinson et al. 2011).

Dans le domaine de l'IHM, l'utilisation de l'IDM est une solution adaptée aux difficultés rencontrées lors de migrations d'un système de contrôle commande, d'une plateforme à une autre (Vernes 2008). Les systèmes de contrôle-commande sont orientés sur une couche à deux niveaux. Au niveau 1, les automates gèrent eux-mêmes leurs programmes. Sur le niveau supérieur, la supervision utilise les données des automates pour animer des synoptiques, des tableaux, etc. L'utilisation de l'IDM pour la conception d'un système de contrôle-commandes revient donc à générer l'interface de contrôle et les codes de commandes correspondant à cette interface.

L'outil SCADA CAD et le projet Anaxagore sont deux approches qui utilisent l'IDM pour générer automatiquement et conjointement les IHM et les codes de commande associés, tout en réduisant le temps et les erreurs de conception.

Contrairement à Anaxagore, l'outil SCADA CAD utilise un système expert, sans prendre en compte les facteurs humains dans le processus de génération. En effet, le processus de génération avec SCADA CAD se fait avec un moteur d'inférence qui s'appuie sur une base de connaissance contenant un ensemble de règles pour gérer la façon dont les informations sont présentées à l'opérateur à travers l'IHM.

Comme dans SCADA CAD, Anaxagore permet de réduire l'effort des intervenants et le temps de conception du projet. Cependant, alors qu'Anaxagore utilise un modèle métier déjà existant connu des experts du domaine, SCADA CAD offre une interface pour créer un modèle d'entrée avec un Domain Specific Language (DSL) (Van Deursen et Klint 2002) qui n'est habituellement pas utilisé dans la conception des systèmes de contrôle-commande. Les connaissances métiers indispensables à la bonne prise en compte de la sémantique du système sont gérées par *Anaxagore* pour produire toutes les données permettant à un logiciel spécialisé de type SCADA de piloter la supervision.

Pour la prise en compte de l'axe fonctionnel dans les systèmes de contrôle-commande, SCADA CAD demande à l'utilisateur du système expert de renseigner toutes les séquences d'actions des fonctions du système, pour permettre le lancement des commandes de haut-niveau, *Anaxagore* permet, à ce jour, de générer des applications de contrôle-commande de bas-niveau. En effet, l'interface de supervision générée par *Anaxagore* ne permet que des commandes

élémentaires du type « ouvrir » ou « fermer » un élément. Cependant, les tâches d'un opérateur de supervision sont plus complexes que celles que nous avons présentées. La réalisation des fonctions de haut niveau à partir de cette interface conduit à des interactions, élément par élément, entre le système et l'utilisateur (commandes de bas-niveau).

À la lumière des types de données d'entrée dont l'outil SCADA CAD a besoin pour la génération des applications de contrôle-commande, l'expert en charge de la description de ces données doit avoir une bonne connaissance du système et doit pouvoir décrire le système conformément au modèle d'entrée qui est exploité par le système expert. Pour chaque élément du système à générer, l'expert système doit renseigner les informations relatives aux blocs de commande, aux capteurs, aux actionneurs et à la structure du système à générer (Bara, Popescu, et Filip 2013) mais également l'ensemble des séquences de commande pour réaliser les fonctions du système à concevoir. Une interface leur est offerte pour la description de ces données. Cependant, en fonction de la complexité du système, la description de toutes ces informations peut être fastidieuse.

1.4 Problématique liée à la spécification

Bien que l'IDM soit fructueuse pour la conception des systèmes de contrôle-commande, elle ne résout qu'une partie du problème car les véritables difficultés sont avant la conception, au moment où l'on cherche à spécifier le système (Gervais 2004). L'importance de cette phase de spécification est particulièrement évidente, quelle que soit la démarche choisie.

Les travaux présentés dans cette thèse se situent dans la conception des systèmes sociotechniques complexes tels que les systèmes de contrôle-commande. Le développement de ces systèmes est un défi dans l'industrie, car il est difficile de comprendre, de tester et de maintenir de tels systèmes. Pourtant, ils nécessitent un développement rapide, de qualité et fiable car ils sont présents dans tous les secteurs de l'industrie.

Certaines approches basées sur l'IDM ont été proposées pour diminuer le temps, les erreurs et les coûts de développement des systèmes de contrôle-commande. Cependant, ces approches ne tiennent pas compte de la spécification fonctionnelle (comme par exemple le projet Anaxagore) et/ou proposent une démarche de spécification trop fastidieuse (l'outil SCADA CAD). De plus, aucune de ces deux approches ne tient compte de l'analyse de la tâche de haut-niveau permettant de faciliter le pilotage d'un tel système. On remarque donc toujours des erreurs de conception généralement dues aux spécifications incomplètes ou imprécises (Standish Group 2014). La prise en compte, dans la conception du système de contrôle-commande, de l'activité de l'utilisateur pour la description des tâches de haut niveau et la réaction (la partie fonctionnelle du système) du système à ces tâches, devient donc essentielle. Sur une IHM de supervision, ces spécifications sont mises en œuvre à travers les commandes de haut-niveau. Une commande de haut-niveau est composée d'interfaces de contrôle permettant à un superviseur de lancer une fonction de haut-niveau et de codes de commandes qui répondent aux actions de haut-niveau du superviseur.

La Figure 18 décrit les activités de conception de système de contrôle-commande de haut niveau. Généralement, l'analyse des tâches humaines est à charge d'un cogniticien qui travaille avec l'opérateur. Le cogniticien communique ensuite son analyse aux concepteurs de l'interface de supervision (informaticiens) pour une modélisation de cette dernière. Les modèles de tâches devront être vérifiés et validés avec l'expert (mécanicien). Quant à la description des spécifications fonctionnelles, elle est à la charge de l'expert (mécanicien), qui

les écrit typiquement en langage naturel, les communique ensuite aux concepteurs de l'interface de supervision (informaticiens) et du programme de commande (automaticiens) qui sont en charge de les implémenter et de les intégrer au système. Les concepteurs d'interfaces de supervision devront tenir compte des modèles de tâches préalablement obtenus. Le programme de commande et l'interface de supervision devront être cohérents, répondre aux exigences du cahier des charges et permettront de piloter le système physique.

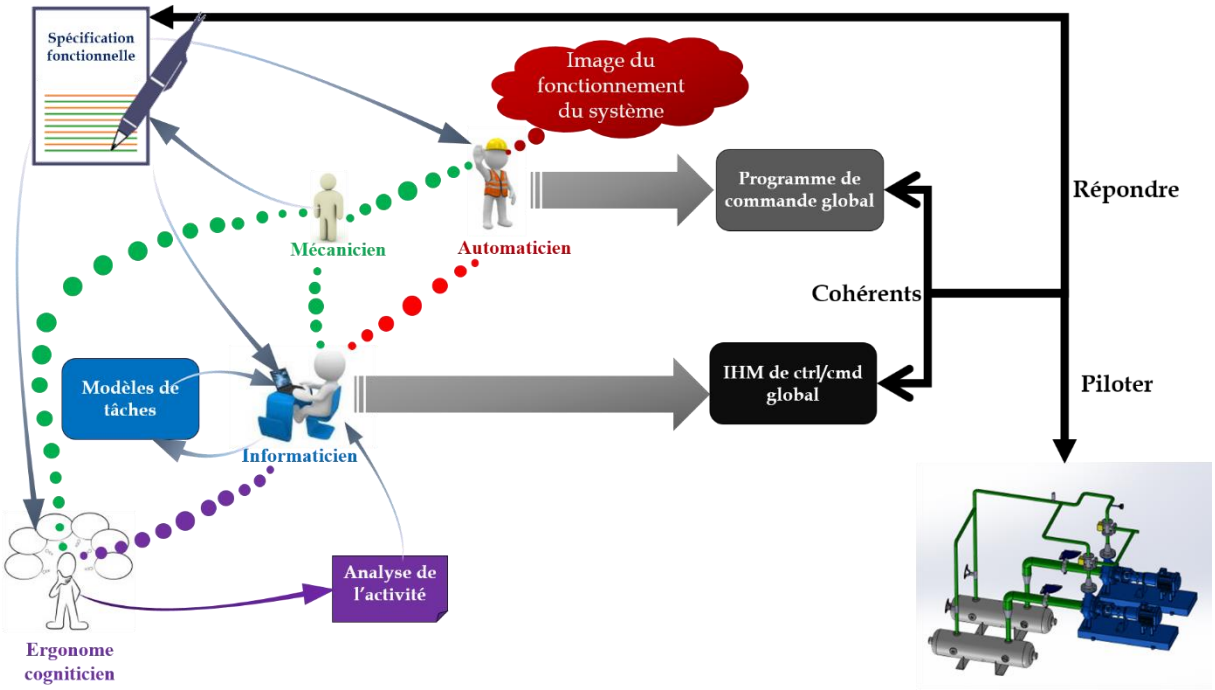


Figure 18 Conception d'IHM et de programme de commande globaux

Les erreurs qui découlent de l'interprétation des spécifications sont dues à la différence de culture technique entre le prescripteur (mécanicien) et les concepteurs (automaticien et/ou informaticien). Ces erreurs affectent le coût de développement et d'évolution des applications. En effet, 55% (Figure 19) des erreurs détectées dans le cycle de vie d'un logiciel proviennent des spécifications (Pham 2006). De plus, c'est pendant la phase de codage qu'on se rend souvent compte que les spécifications initiales étaient incomplètes, incohérentes, fausses ou irréalisables. Ces découvertes tardives de défauts affectent le coût de développement et d'évolution des applications.

Pour avoir un bon système « *du premier coup* », il faut donc s'assurer que les spécifications du système soient complètes et correspondent bien aux besoins du client. Cependant, il est quelquefois nécessaire de prendre en compte des changements importants dans les spécifications dans une phase avancée du projet. Ces changements risquent d'affecter le système de sorte qu'il ne correspond plus aux besoins initiaux qui évoluent dans le temps.

Dans le contexte actuel du «*plus complexe* », «*premier coût* », « *plus vite* » et « *moins cher*», les concepteurs de système de contrôle-commande doivent relever plusieurs défis :

- Définir des spécifications plus tôt, plus facilement et plus sûres.
- Vérifier et valider les spécifications au plus tôt,
- Tenir les délais même en cas de grosses modifications dans les spécifications, et ce, même dans une phase avancée du projet,

- Concevoir plus vite les modèles de tâches qui sont complexes,
- Réduire le temps de conception général,
- Promouvoir la réutilisation.

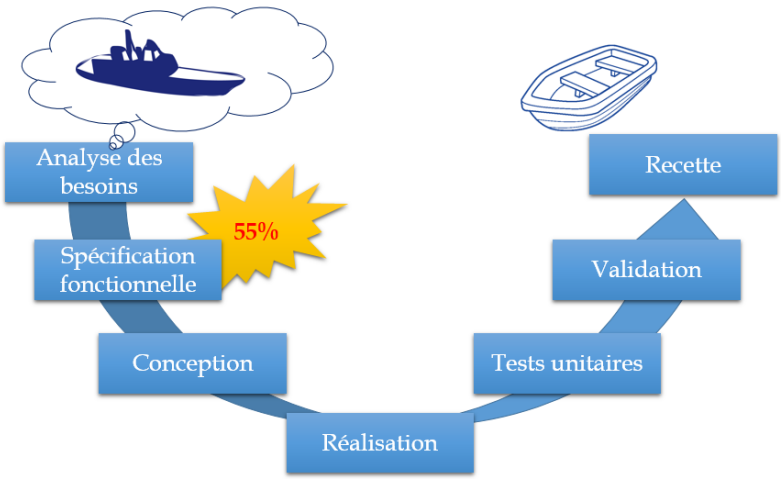


Figure 19 Origine des erreurs de conception

Ces défis amènent vers deux grandes questions qui sont au cœur de ce mémoire :

1. Comment aider les concepteurs à spécifier plus facilement l'axe fonctionnel du système ?
2. Comment aider les concepteurs à obtenir plus facilement les modèles de tâches issus de l'analyse de la tâche humaine, surtout complexe, tout en promouvant la réutilisabilité ?

Aussi, l'intégration des spécifications dans le processus général de génération d'un système de contrôle-commande, afin de réduire les temps de conception et de re-conception est une étape importante.

L'activité de spécification (fonctionnelle, structurelle, tâche interactive de supervision, etc.) permet de modéliser le système à concevoir en s'appuyant sur des notations et formalismes. (BARTHET 1988) a défini le formalisme comme une convention de représentation des concepts de modèle. Dans cette section, nous passons en revue quelques formalismes existants permettant de décrire le fonctionnement du procédé et la tâche de l'opérateur. Cette thèse s'intéresse à l'activité de spécification fonctionnelle de haut niveau et de la tâche interactive de supervision qui permettront de générer le système final.

2 Formalisme de spécification pour la construction des systèmes de contrôle-commande

Généralement, pour construire les interfaces de supervision, le procédé physique est déjà existant ou du moins l'analyse fonctionnelle et/ou structurelle est déjà réalisée. La plupart du temps, les besoins pour la conception d'une application sont exprimés par le client dans le cahier des charges rédigé en langage naturel. De ce cahier des charges découle un dossier d'analyse ou de spécifications fonctionnelles. La spécification est une base de communication entre le client et les équipes de conception (Clarke et Wing 1996). Les documents de spécifications doivent être le plus explicite possible car ils constituent un point de référence sur lequel les informaticiens s'appuient pour le développement de l'axe fonctionnel du système (IEEE Compunter Society 1998). Une bonne spécification fonctionnelle doit être

correcte, sans ambigüité, complète et cohérente. Certaines méthodes formelles ou semi-formelles ont été proposées pour aider au respect de ces propriétés. Les méthodes formelles permettent l'expression des spécifications avec des notations et sémantiques basées sur des concepts mathématiques. Ces méthodes se distinguent en trois principales classes : les langages de spécification basés sur les modèles, les langages de spécification algébrique et les langages de spécification graphique. Dans chaque classe de langages, nous nous intéressons aux approches proposées pour la spécification fonctionnelle du système pour tenter de répondre à notre problématique.

Notre état de l'art est complété par un retour d'expérience basé sur les documents de spécification fonctionnelle utilisés dans le cadre de projets réels.

2.1 La spécification basée sur les modèles

La spécification basée sur les modèles est une approche de spécification formelle qui permet d'exprimer la spécification du système sous forme de modèle d'état. Ce modèle d'état est construit en utilisant des entités mathématiques telles que les ensembles et les fonctions. Les opérations du système sont décrites en définissant la façon dont elles affectent l'état du modèle. Les langages de spécification basés sur les modèles sont utilisés pour résoudre les ambiguïtés dans les spécifications ou détecter les erreurs de conception au début du cycle de vie du projet (Jalila et Mala 2014). Il existe plusieurs notations de spécification basées sur les modèles dont les plus utilisées sont : Z (Spivey 1989), B (ABRIAL 1996) et VDM (Vienna Development Method) (Jones 1980).

2.1.1 La spécification avec Z

Dans les années 80, le groupe de recherche de l'université d'Oxford a développé la notation Z (Spivey 1989) qui est reconnue comme une norme internationale ISO/IEC JTC1 SC22. La notation Z s'appuie sur la théorie des ensembles et sur la logique des prédicats du premier ordre. Cette notation est souvent utilisée pour la spécification de systèmes critiques où la réduction des erreurs et la qualité du logiciel sont extrêmement importantes (Pandey et Srivastava 2015). Une spécification en Z permet d'utiliser les théories mathématiques des ensembles, relations et fonctions pour construire explicitement un modèle mathématique. Z ne permet de spécifier que les aspects fonctionnels d'un système (Jalila et Mala 2014).

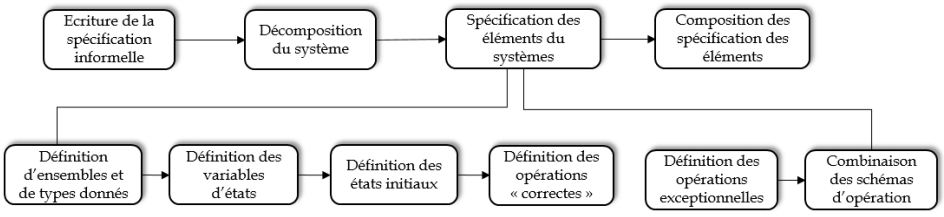


Figure 20 Processus de développement d'une spécification avec Z

Le processus de spécification de Z décrit une approche (Figure 20) qui permet de décomposer le système en éléments pour le spécifier. Les fragments de spécification obtenus sont ensuite regroupés pour former la spécification complète. À la fin du processus, on obtient une collection de schémas qui peuvent être combinés et utilisés dans d'autres schémas. Un schéma est une représentation de la notation pour l'affichage des prédicats qui sont utilisés dans la définition des opérations et des invariants ; et se compose d'une déclaration et d'une liste opérationnelle des prédicats. La sémantique d'une spécification Z est composée d'un ensemble

de variables globales, d'un certain nombre d'invariants pour ces variables ainsi que de pré et post-conditions pour les opérations d'observation ou de modifications de variables globales. Cette sémantique repose sur une «boîte à outils», plutôt complexe, de constructions mathématiques. Pour la spécification des grands systèmes, la notation Z est un outil professionnel puissant. Cependant, deux défis sont à relever dans l'utilisation de la notation Z : l'apprentissage d'un Framework syntaxique relativement simple et l'apprentissage d'une grande bibliothèque de notations mathématiques.

Plusieurs outils ont été proposés pour supporter la spécification formelle en Z. Z/EVES (Pandey et Srivastava 2015) est l'un des outils les plus avancés mais n'est pas très simple d'utilisation.

2.1.2 La spécification avec B

Inventée dans les années 1980 par Jean-Raymond Abrial (ABRIAL 1996), la méthode B est une méthode de spécification formelle qui permet une description très rigoureuse des exigences exprimées dans un cahier des charges, afin de garantir la fiabilité et la sûreté des applications à concevoir. Avec la méthode B, l'ensemble du processus de spécification, de conception et de codage est décrit de façon uniforme (Aït-Ameur, Girard, et Jambon 1998) et s'appuient sur la réalisation un certain nombre de preuves mathématiques.

Le processus de spécification avec B (Figure 21) commence par une modélisation abstraite du comportement et des spécifications d'un système dans le langage B, puis des raffinements successifs permettant de rajouter des détails au modèle du système jusqu'à obtenir un modèle concret. Une activité de preuve formelle est utilisée tout au long du processus de spécification pour vérifier la cohérence avec le modèle abstrait de départ et la conformité de l'ensemble des raffinements avec le modèle abstrait. Le modèle concret issu de la spécification est automatiquement traduit en langage informatique.

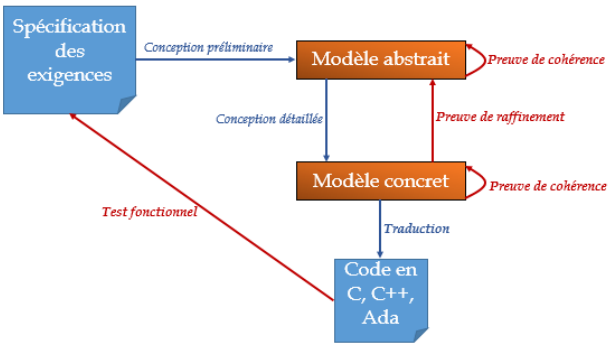


Figure 21 Processus de spécification avec B

Bien que la méthode B permette d'obtenir des spécifications techniques structurées, cohérentes et sans ambiguïté, elle conduit à l'utilisation d'un formalisme mathématique de haut-niveau.

2.1.3 La spécification avec VDM

VDM (Vienna Development Method) (Jones 1980) a été développé dans les laboratoires d'IBM de Vienne. Les spécifications en VDM sont basées sur des assertions logiques des états abstraits (abstraction mathématique et spécification d'interface). Contrairement à Z, VDM utilise des mots clés pour distinguer les rôles des différents composants alors que ces structures ne sont pas explicites dans Z. Comme les spécifications en Z, les spécifications avec VDM ne sont généralement pas exécutables. Une spécification en VDM décrit une machine à états qui est

composée d'un ensemble d'états et d'une collection d'opérations. Ces opérations changent l'état du modèle.

VDM supporte plusieurs types de données tels que : les ensembles, les listes, les mappings et les opérations. Cette méthode ne permet de spécifier que les aspects fonctionnels d'un système (Jalila et Mala 2014).

Une extension de VDM a été proposée : VDM++ (Jalila et Mala 2014) qui est une méthode de spécification orientée objet permettant de représenter le système sous forme d'un ensemble de classes. Ces classes peuvent encapsuler des constantes, des variables, des opérations et des fonctions et décrivent l'état du système. VDM++ est particulièrement adaptée à la spécification des systèmes temps réels distribués et des systèmes de contrôle-commande car elle offre des *threads* pour gérer les propriétés de concurrence.

2.2 La spécification algébrique

En génie logiciel, la spécification algébrique est la première étape de la production d'un logiciel qui permet de spécifier formellement les fonctionnalités attendues du système. Cette formalisation permet de décrire le logiciel avec une signature composée de types de données et des opérations mathématiques sur ces types de données. La logique sous-jacente est de permettre de prouver les propriétés des spécifications pour obtenir des programmes plus efficaces. Il existe plusieurs langages de spécification algébrique recensés dans (Wirsing 1995). Parmi eux se trouvent les langages : SPECTRAL (Krieg-Brückner 1990), SPECTRUM (Broy, Facchi, et Grosu 1993) et EML (Sannella et Torlecki 1985) qui sont spécifiques à la description de la spécification fonctionnelle du système, par l'utilisation d'une logique d'ordre supérieur. Nous avons choisi ici de présenter SPECTRAL et SPECTRUM qui sont deux des méthodes algébriques utilisées pour la spécification fonctionnelle des systèmes.

2.2.1 Le langage SPECTRAL

SPECTRAL (Krieg-Brückner 1990) est un langage de spécification qui inclut des fonctions partielles d'ordre supérieur, des spécialisations d'ordre supérieur par *types dépendants*, des sous-classes et une forme d'héritage. Ce langage offre des caractéristiques qui permettent de représenter le système à deux différents niveaux d'abstraction : les spécifications abstraites des exigences et les spécifications de conception exécutables (Wirsing 1995).

SPECTRAL combine une généralisation et une extension des approches EML (Sannella et Torlecki 1985) pour les aspects de développement nommés « *in-the-large* » et PROSTECTRA (Krieg-Brückner 1990) pour les aspects de développement nommés « *in-the-small* ». Pour la spécification « *in the small* » le problème est de choisir le type de signature dans lequel les axiomes du système logique seront écrits; et une notation algébrique pour définir le sens des axiomes via la définition de la satisfaction. Pour la spécification « *in the large* », les langages de spécification fournissent des mécanismes pour construire les systèmes complexes de façon modulaire à partir de petites unités. Ces mécanismes diffèrent naturellement de ceux utilisés dans la spécification « in the Small ». En effet, les « petits » objets (valeurs) sont différents des «grands» objets qui représentent les structures correspondant à de nombreuses classes algébriques. La structuration est importante pour l'analyse, la compréhension et la description des grandes spécifications. Les concepts de structuration des langages de spécification sont définis dans (Krieg-Brückner 1990).

Les spécifications sont uniformisées avec une notion de *type* et de *sous-type* pour les petits et les grands objets (type de valeur et classe de structure) ainsi qu'une notion générale de fonction d'ordre supérieur entre les quatre éléments suivants : les valeurs, les types, les structures et les classes. Les structures sont des algèbres et les spécifications (classes) représentent un ensemble d'algèbres. Faire une distinction entre les algèbres et les spécifications, en fournissant des fonctions d'ordre supérieur sur ceux-ci, donne un cadre très général pour exprimer la structuration des spécifications d'une part, et d'autre part, la structuration de l'implémentation des projets paramétrés (Sannella et Tarlecki 1988). Les notations de *type* permettent l'expression de l'héritage comme une nouvelle manière de structurer. Cela répond aux besoins liés aux différentes façons de structurer les spécifications généralisées (en particulier à la phase de description des exigences) et les implémentations génériques d'un système logiciel particulier.

2.2.2 Le langage SPECTRUM

Le langage SPECTRUM (Broy, Facchi, et Grosu 1993) est basé sur des techniques de spécification axiomatiques et est orienté vers des programmes fonctionnels. Ce langage permet d'implémenter des spécifications exécutables à partir du cahier des charges et supporte explicitement les fonctions partielles, continues et d'ordre supérieur. Il supporte également les polymorphismes et les objets infinis (Wirsing 1995) et est composé d'un langage de spécification, d'un calcul de la déduction (*deduction calculus*) et d'une méthodologie de développement.

Le processus de développement des spécifications avec SPECTRUM permet d'obtenir des spécifications précises bien structurées. Comme son nom l'indique, un large éventail de styles de spécification est envisagé, y compris des moyens pour décrire les systèmes concurrents. SPECTRUM permet à l'utilisateur d'écrire des spécifications très abstraites (non exécutables), ainsi que des spécifications qui peuvent être lues et exécutées sous forme de programmes fonctionnels. La sémantique d'une spécification décrite avec SPECTRUM est représentée par la classe de tous les algèbres avec la signature appropriée qui satisfait les axiomes.

2.3 La spécification graphique

Les approches traditionnelles de l'analyse fonctionnelle utilisent les diagrammes de flux pour modéliser les blocs fonctionnels des systèmes de contrôle-commande (Fiorèse et Meinadier 2012). Dans la littérature nous avons quatre types de diagramme de flux. Il s'agit des diagramme de flux mécanique (*Mechanical Flow Diagram (MFD)*) (Richard Turton, Richard C. Bailie, Wallace B. Whiting 2008), des diagrammes de données (*Data Flow Diagrams (DFD)*) (Le Vie et Donald 2000) qui permettent de modéliser la structure d'un système, des diagrammes de flux de commande (*Control Flow Diagrams (CFD)*) (McAvinew et Mulley 2004) et des diagrammes de flux de fonctions (*Functional Flow Bloc Diagrams (FFBD)*) (McAvinew et Mulley 2004) qui permettent de modéliser le comportement dynamique d'un système . Ces deux types d'approches sont connues sous le nom d'approche structurelle (DFD) et approches comportementale (CFD et FFBD) (Fiorèse et Meinadier 2012). Dans la conception des systèmes de contrôle-commande, l'approche structurelle est souvent utilisée pour construire des schémas comme le P&ID (Pipping and Intrumentation Diagram) (Richard Turton, Richard C. Bailie, Wallace B. Whiting 2008) connu sous le nom de MFD et l'approche comportementale est souvent mise en œuvre à travers le SADT (Structured Analysis and Design Technique). Ces deux approches se concentrent sur la description fonctionnelle du système. Elles sont alors

complétées par les modèles de tâches, généralement décrits par les concepteurs d'IHM, qui permettent de décrire les actions de l'utilisateur sur le système.

2.3.1 Spécification graphique du point de vue du système

2.3.1.1 Le P&ID

Le P&ID est une description graphique détaillée du procédé physique suivant la norme ANSI/ISA-S5-1 (Ansi/Isa 1992). C'est un schéma structuro-fonctionnel normalisé utilisé au plus tôt dans les projets de conception de système de contrôle-commande car il offre une vue d'ensemble du système et une représentation détaillée de ses composants.

La description du procédé à travers un P&ID se base sur des symboles prédéfinis simples et facile à comprendre par tous les acteurs engagés dans la conception du système de contrôle-commande. Cette description fournit aux ingénieurs des informations nécessaires pour la conception du procédé, concernant toute la tuyauterie, l'équipement, et une grande partie de l'instrumentation.

Généralement, le P&ID sert de guide aux intervenants du projet. Par exemple, il permet aux ingénieurs mécaniciens et civils de construire et d'installer les équipements ; aux ingénieurs d'instrumentation (responsable de la conception, du développement, de l'installation, de la gestion et/ou maintenance des modules qui sont utilisés pour surveiller et contrôler les procédés) de spécifier, d'installer et de vérifier le système de contrôle-commande, etc. De plus, le P&ID sert de base pour former les opérateurs sur ce qu'ils peuvent faire sur le procédé physique et sur comment utiliser l'interface de supervision associée (Richard Turton, Richard C. Bailie, Wallace B. Whiting 2008).

Formellement vérifié (Mesli-kesraoui, Kesraoui, et al. 2016), le PID est considéré comme une spécification graphique formelle. Cette vérification formelle est nécessaire car il est généralement construit par l'ingénieur de procédé (expert métier) (McAvinew et Mulley 2004) et peut contenir des erreurs. Il est souvent complété par des CFD qui décrivent les flux liés au pilotage du système (Fiorèse et Meinadier 2012).

2.3.1.2 SADT

SADT (D. Marca et McGowan 1988) fait partie des approches traditionnelles de l'analyse fonctionnelle qui s'appuient sur les techniques de décomposition et sur les représentations des flots de données pour donner l'arborescence des fonctions d'un système. Dans cette approche, le système de contrôle-commande est identifié à une fonction globale qui est ensuite décomposée en des fonctions plus détaillées. L'analyse fonctionnelle est donc considérée comme descendante car elle part du général pour aller au particulier.

Fréquemment utilisée dans l'industrie, la notation SADT permet une description graphique du système en combinant le DFD et le CFD pour tenir compte des aspects structurels et comportementaux (Fiorèse et Meinadier 2012). Cette méthode reste tout de même inadaptée à la conception des systèmes de contrôle-commande car elle ne permet pas de prendre en compte toute la dynamique du système (MOUSSA 2005) et n'est pas souvent cohérente avec le PID qui est une représentation structurelle du système.

2.3.2 Spécification graphique du point de vue de l'utilisateur : la modélisation des tâches

Les approches de spécification fonctionnelle se placent du point de vue du système (et du concepteur). L'utilisateur n'y est généralement pas pris en compte. Cependant, dans la conception des systèmes de contrôle-commande, la prise en compte de l'utilisateur est incontournable car elle permet d'éviter des erreurs de d'utilisation. L'étape de spécification fonctionnelle est alors souvent complétée par l'analyse et la conception de modèles de tâches.

L'utilisation des modèles de tâches dans une conception pluridisciplinaire est devenue de plus en plus répandue (Lewandowski, Bourguin, et Tarby 2007). Généralement le modèle de tâches constitue l'un des points de départ de la conception car il aide à la compréhension de l'activité humaine et de l'utilisation du système à concevoir. Les principes de base des modèles de tâches ne diffèrent pas beaucoup d'une modélisation de tâches à une autre, quelle que soit l'approche utilisée (Lachaume et al. 2014).

Les modèles de tâches visent à exprimer les tâches humaines en s'appuyant sur l'analyse de l'activité pour répondre aux besoins de la conception. Les tâches sont généralement décrites suivant des mécanismes de structuration. Ces mécanismes offrent des **opérateurs temporels** qui permettent une décomposition hiérarchique des tâches en sous-tâches, avec une catégorisation en fonction des **types de tâches**. En vue d'une description plus précise de l'activité, il est possible de rajouter aux tâches des **attributs** tels que *optionnelle*, *itérative*, ainsi que des *pré* et *post-condition*.

2.3.2.1 Les types de tâches

Les types de tâches permettent de décrire plus aisément l'activité humaine en fournissant des informations plus précises sur la nature de la tâche (Martinie 2011). Pour les activités qui conduisent à des interactions entre le système et l'utilisateur, les types de tâches permettent de distinguer les rôles lors de la description du modèle. Dans les approches récentes de modélisation de tâches telles que CTT (F. Paternò, Mancini, et Meniconi 1997), K-MAD[6], HAMSTERS (Martinie 2011), etc., on retrouve généralement au minimum quatre types de tâches : tâches *système*, tâches *utilisateur*, tâches *interactives* et tâches *abstraites*.

Les *tâches systèmes* concernent les fonctions exécutées uniquement par le système. Elles concernent les traitements informatiques et le retour d'information sur les résultats de ces traitements. Pour les *tâches utilisateur*, seul l'utilisateur est engagé dans la réalisation des tâches. Ces tâches impliquent par exemples des activités de réflexion, de prise de décision, etc. Les *tâches interactives* impliquent à la fois l'utilisateur et le système. Elles sont utilisées lorsqu'il y a une information à passer entre l'utilisateur et le système (l'utilisateur agit et le système répond). Les tâches abstraites sont des tâches complexes qui sont décomposables en sous-tâches de types différents.

2.3.2.2 Les opérateurs

Les opérateurs permettent de décrire la dynamique du modèle de tâches. Dans les approches de modélisation de tâches, il existe plusieurs opérateurs qui permettent d'exprimer la relation entre les sous-tâches d'une tâche décomposée. Ces opérateurs s'appuient sur un sous-ensemble d'opérateurs LOTOS (Fabio Paternò et Faconti 1992). Dans CTT par exemple, l'une des approches initiales de modélisation de tâches, on distingue quatre opérateurs :

[6]http://lisi-forge.ensma.fr/forge/projects/kmade

- L'opérateur *Enabling* >> permet d'exprimer le fait que la réalisation des sous-tâches doit suivre une séquence prédéfinie.
- L'opérateur *Choice* [] permet d'exprimer le fait que l'utilisateur peut choisir entre plusieurs sous-tâches.
- L'opérateur *Concurrent* ||| d'exprimer le fait que les sous-tâches devront être exécutées simultanément.
- L'opérateur *Order independant* |=| permet d'exprimer le fait que les sous-tâches ne sont pas tenues d'être réalisées dans un ordre spécifique.

2.3.2.3 Les attributs et les conditions

Les attributs et les contextes permettent d'avoir une précision dans la description de l'activité. Parmi les attributs ceux qui reviennent souvent et qui ont des conséquences sur la dynamique du modèle (Lachaume et al. 2014) on peut noter l'attribut *itération* et l'attribut *optionnel*. L'itération indique que la tâche est répétitive. Le choix de l'attribut *optionnel* sur une tâche indique que, selon le contexte, la tâche peut ne pas être exécutée.

La précision sur le contexte et les conditions de réalisation d'une tâche est importante pour la description fine de l'activité. Deux types de conditions sont généralement exprimés dans les modèles de tâches. L'expression d'une précondition sur une tâche permet de définir le contexte (les prérequis de la tâche) dans lequel la tâche est réalisable. Les post-conditions, quant à elle, expriment l'état requis après l'action.

2.4 Retour d'expérience industriel sur les spécifications

Pour connaître les informations contenues dans les spécifications fonctionnelles utilisées en industrie, nous avons analysé plusieurs documents de spécification, écrits en langage naturel, qui décrivent les fonctions de haut-niveau de systèmes réels (Système d'eau douce (EdS), système de gazole, système de stockage et de distribution de lubrifiant, carburateur, etc.). Dans ces documents, le lancement de certaines de ces fonctions se fait de façon semi-automatique suivant le même principe et en plusieurs phases. La première est la phase de choix. L'opérateur doit cliquer sur la commande globale représentant la fonction pour afficher l'interface de saisie de consignes lui permettant de poursuivre sa tâche. La deuxième phase est la phase de définition. Au cours de cette phase, il doit faire des vérifications, renseigner la ou les consignes, valider ou annuler sa tâche. La troisième phase dépend de la précédente car elle ne se réalise que lorsque l'opérateur a validé sa consigne. Elle est donc pré-conditionnée. C'est la phase d'initialisation où le système répond à la demande de l'opérateur soit négativement soit positivement. Enfin, vient la quatrième phase qui est la phase de contrôle pendant laquelle l'opérateur fait de la surveillance des fonctions et agit en fonction des informations du système et de ses besoins.

Dans les documents de spécification fonctionnelle étudiés, la démarche ci-dessus est décrite pour chacune des fonctions de haut-niveau du système à concevoir car les consignes à renseigner ainsi que leur nombre et les alertes que le système peut renvoyer diffèrent d'une fonction à une autre.

2.5 Synthèse sur la spécification

Afin de proposer une démarche permettant d'obtenir des spécifications fonctionnelles « sûres », nous avons choisi de présenter dans cette section, quelques principales approches formelles utilisées pour la spécification fonctionnelle des systèmes.

Les approches telles que Z, B et VDM sont utilisés pour spécifier les aspects fonctionnels du système sous forme de modèles, de façon déclarative en utilisant des contraintes. L'utilisation de ces méthodes se fait par des notations mathématiques plus ou moins complexes. En effet, Z utilise beaucoup de notations mathématiques pour décrire les aspects statiques et dynamiques du système tandis que VDM utilise peu de symboles mathématiques. Toutefois, VDM++, une extension de VDM basée sur une approche orientée objet, dispose d'une sémantique et d'une syntaxe facilement compréhensible. Dans VDM++ le système est représenté par un ensemble de classes et chaque classe décrit un état du système. Face à des systèmes complexes, le nombre de classes peut devenir très conséquent. Contrairement à Z, VDM/VDM++ met l'accent sur la gestion des exceptions et des propriétés de concurrence. Il est donc plus adapté à la spécification fonctionnelle des systèmes de contrôle-commande. Cependant, avec VDM, les spécifications peuvent contenir des défauts fréquents et conduire à des coûts plus élevés en développement (Jalila et Mala 2014). En effet, les méthodes formelles ne garantissent pas la liberté d'erreur (elles ont souvent été «survendues») (Ciapessoni et al. 1999). Pour chaque langage de spécification, il existe des outils permettant leur mise en œuvre, mais ils ne facilitent pas pour autant toujours la tâche de spécification. Par exemple, VMTools, un des outils associés à VDM, manque d'utilisabilité. Il n'y a pas d'éditeur interne pour les modèles, l'utilisateur doit toujours utiliser, par exemple Microsoft Word, pour changer la spécification. De plus, la liste des erreurs ne peut pas être vidée et il est donc difficile de différencier les nouvelles erreurs des anciennes (Müller 2009).

La méthode B, quant à elle, dispose également d'outils issus des ateliers de génie logiciel « L'atelier B[7] » et « BToolkit[8] » qui permettent d'assister et de contrôler les tâches de spécification. Contrairement à Z et VDM, la méthode B ne se limite pas à la phase de description des spécifications (Aït-Ameur, Girard, et Jambon 1998). En effet, la méthode B propose une chaine complète de conception allant de la spécification du programme au code source associé. Cependant, les utilisateurs de cette méthode rencontrent une réelle difficulté dans la lecture et la manipulation des données (MOUSSA 2005), et doivent suivre une formation très poussée pour être efficaces dans leur utilisation.

Les approches de spécification algébrique telles que SPECTRAL et SPECTRUM se caractérisent par leur syntaxe, leur sémantique et leur calcul de la preuve associée. Les différences entre ces langages viennent de différents facteurs tels que leur utilisation prévue dans le processus de développement de logiciels, le paradigme de programmation utilisé et le fait qu'ils offrent ou pas un outil (Wirsing 1995). Ces deux langages permettent de décrire les spécifications fonctionnelles d'un système. Dans SPECTRAL, les dependent types sont utilisés pour tous les objets. Les avantages des dependent types sont les conditions de contexte qui peuvent être contrôlées au niveau des signatures et, plus important encore, les mécanismes logiques et structurels sont intégrés dans le même Framework. Un inconvénient est que toutes les dependent type sont rendus explicites, conduisant à des expressions avec de nombreux paramètres qui peuvent être difficiles à lire. De plus, la vérification des types statiques ne peut pas être réalisée en présence de sous-types définis par des prédicats. SPECTRUM, quant à lui, ne dispose d'aucune notion d'intégration d'un état de programme, et tout le style de spécification est orienté vers la programmation fonctionnelle. Un certain nombre de concepts

[7] http://www.atelierb.eu/
[8] http://www.b-core.com/cgi-sys/suspendedpage.cgi

linguistiques fonctionnels ont donc été intégrés dans ce langage pour fournir le polymorphisme paramétrique dans le style du langage de programmation fonctionnel.

Globalement, il ressort de notre analyse que les langages de spécification formels permettent une analyse rigoureuse des spécifications d'un système. Ces langages sont basés sur des concepts mathématiques et leur utilisation permet de réduire, au plus tôt dans le cycle de développement du projet, le coût de développement et les défauts qui peuvent subvenir dans le système opérationnel. Il existe plusieurs langages de spécification formels. Le choix du langage dépend du type de système à concevoir. Cependant, l'utilisation de ce langage nécessite une formation aux concepts mathématiques car ces concepts ne sont pas connus par les spécificateurs des systèmes de contrôle-commande.

Les approches de spécification graphique sont beaucoup plus faciles à aborder. Celles qui sont le plus utilisées dans l'industrie, telles que le P&ID et SADT, se caractérisent par une description du système permettant de faciliter le dialogue entre les équipes de conception. Par exemple, le PID permet d'avoir une idée générale pour contrôler le système mais ne contient pas toutes les informations nécessaires à la conception de l'axe fonctionnel du système de contrôle-commande (McAvinew et Mulley 2004). Il ne permet de décrire que la vision structurelle du système. Il est donc souvent complété par d'autres diagrammes (Richard Turton, Richard C. Bailie, Wallace B. Whiting 2008). Dans certains secteurs de l'industrie, SADT est souvent utilisé pour la prise en compte de la vision comportementale du système. Cependant, cette méthode de spécification n'est pas totalement adaptée car d'une part peut nécessiter un codage assez lourd et fastidieux en fonction de la complexité du système (Demuynck et Meyer 1979), et d'autre part elle ne contient pas toutes les informations sur le fonctionnement du système. De fait, elle est souvent complétée par un document exprimé en langage naturel pour compléter les informations manquantes pour la spécification du système ou par des modèles exprimés en RdPI (Réseaux de Petri Interprétés) par exemple (Abed 2001).

Les méthodes de spécification décrites jusque-là, sont centrées système alors que l'activité de supervision nécessite une intervention importante de l'utilisateur, qui doit pouvoir utiliser efficacement le système en limitant les risques d'erreur. La prise en compte des activités de l'utilisateur nous a conduits à considérer les modèles de tâches, qui, par ailleurs, permettent de générer des parties de systèmes. L'analyse de certains des documents utilisés dans l'industrie, nous a permis recueillir certaines informations pouvant servir à la conception des modèles de tâches.

Plusieurs outils basés sur différentes notations fournissent des mécanismes de structuration qui exploitent les principes de base des modèles de tâches, pour faciliter la description de l'activité humaine. Ces outils proposent systématiquement un simulateur de tâches pour vérifier et valider les modèles de tâches conçus.

3 Verrous identifiés par l'état de l'art

Les spécifications fonctionnelles décrivent la façon dont un système permet d'atteindre les buts de l'utilisateur et contiennent notamment une liste des principales fonctions du système (exigences de haut niveau) et des scénarios d'exploitation. L'expression claire des besoins et des exigences, et leur traduction en spécifications fonctionnelles, n'est cependant pas chose aisée, malgré l'importance de ces dernières pour la phase de conception (Kolski et Ezzedine 2003).

La plupart du temps, la description des exigences pour établir les spécifications fonctionnelles, est la première étape dans le développement de systèmes de contrôle-commande. Dans cette section, nous avons décrit les langages de spécification formels généralement utilisés pour la description des spécifications fonctionnelles de système. Ces langages sont utilisés pour réduire le coût de développement et les défauts dans le système opérationnel (Jalila et Mala 2014). Ils permettent de diminuer l'ambiguïté et, garantissent l'exhaustivité et l'exactitude des spécifications (Pandey et Srivastava 2015). En effet, l'analyse fonctionnelle et/ou structurelle d'une installation, sans l'utilisation d'un ou de plusieurs formalismes, s'avère souvent fastidieuse en raison de la complexité croissante des procédés (MOUSSA 2005). Cependant, dans certains secteurs de l'industrie, comme la construction navale, le langage naturel (Wiegers et Beatty 2013) reste privilégié car les spécificateurs n'ont pas la culture technique nécessaire à l'expression des spécifications dans d'autres langages. En effet, avoir un bon formalisme pour spécifier, analyser, implémenter, et vérifier les applications informatiques, même s'il est bien outillé, ne suffit pas si les utilisateurs de ce formalisme ne sont pas formés à leur utilisation (Ciapessoni et al. 1999).

Généralement, la conception des applications de contrôle-commande est difficile car beaucoup d'erreurs dans ces applications peuvent être dues à des défauts dans leurs spécifications. Bien qu'il existe des techniques de détections de défauts dans les spécifications (Fagan 1976; Singh 1996), elles nécessitent beaucoup d'efforts et ne permettent pas forcément de tout détecter. Malgré l'utilisation de certaines de ces techniques, on retrouve des erreurs non détectées dans de nombreux documents de spécifications (Trudel 2012).

La spécification de ce qu'un système doit faire précisément, reste un aspect principal dans le développement de l'interface utilisateur du système. La génération automatique des interfaces est un problème difficile car il nécessite de déterminer la spécification exacte de ces dernières. La question que l'on se pose est : comment formaliser les spécifications et construire un générateur d'interface qui peut concevoir une interface utilisable à partir de ces spécifications.

Dans la construction navale, le modèle métier est considéré comme une spécification formelle de bas-niveau, car c'est un schéma structuro-fonctionnel standardisé qui est formellement vérifié (Mesli-kesraoui, Kesraoui, et al. 2016). Ce schéma est utilisé systématiquement et plus tôt dans la conception. Il suit tout le cycle de vie de conception du système (Bignon 2012). Cependant, cette description ne suffit pas à implémenter des fonctions de haut-niveau du système. Pour générer un système global, il est important de connaître les informations fonctionnelles nécessaires à la création d'une interface utilisable et de ses codes de commande. Les experts métier (mécaniciens) ont cette connaissance du système, mais ils ne disposent pas de formation pour les exprimer dans les langages formels. Ils ne peuvent les décrire qu'en langage naturel en utilisant le modèle métier. La question que nous nous sommes posée est : comment pouvons-nous les aider à exprimer correctement les spécifications ?

L'un de nos objectifs est donc de faciliter la spécification fonctionnelle des systèmes complexes en proposant une démarche permettant de capturer de façon rigoureuse l'expertise d'un expert métier, sur les aspects fonctionnels et les contraintes d'usage des systèmes de contrôle-commande, tout en tenant compte des critères et recommandations ergonomiques.

Par ailleurs, dans la démarche de conception des systèmes interactifs, la phase de spécification fonctionnelle est souvent précédée de l'analyse de la tâche humaine. Une tâche amène à un but (état du système voulu) que l'utilisateur souhaite atteindre en utilisant une procédure qui

décrit les moyens pour atteindre ce but (Normand 1992). L'analyse de la tâche permet d'avoir un ensemble de données qui peuvent être exploitées dans les approches de conception. Il est aujourd'hui admis que les modèles de tâches sont totalement intégrés dans le processus de conception des systèmes interactifs. Face à la complexité des systèmes interactifs, plus spécialement des systèmes de contrôle-commande, où les tâches de l'utilisateur doivent être précises et tenir compte du contexte, la conception des modèles de tâches en devient plus fastidieuse. En effet, avec les actions à définir, les états du système à surveiller (les alarmes, les éléments, les défauts etc.), les actions possibles en fonction du contexte, les prises de décision etc., la conception du modèle de tâche est fastidieuse. De plus, la détection et la correction des erreurs lors de la simulation de ces modèles peuvent retarder la conception du système de contrôle-commande. En effet, les modèles de tâches sont généralement conçus par le concepteur d'IHM puis simulés avec l'expert métier jusqu'à leur validation.

Il apparaît alors intéressant de rechercher des pistes permettant d'une part de faciliter la conception des modèles de tâches complexes, et d'autre part de capturer les connaissances de l'expert métier pour obtenir plus facilement ces modèles.

Chapitre 2 : Faciliter la conception des modèles de tâches complexes

Dans la conception des systèmes de contrôle-commande, les modèles de tâches permettent de décrire les actions que l'opérateur en supervision fait pour lancer, contrôler, surveiller les évènements du système (les alertes, les messages de bon fonctionnement), évaluer, arrêter une fonction (s'il y a besoin), etc. Selon l'état du système supervisé (présence de défauts par exemple), ces tâches peuvent être nombreuses et compliquées. Les modèles de tâches permettant de les décrire sont donc complexes.

Dans ce chapitre, nous tentons d'apporter une solution aux difficultés rencontrées dans la conception des modèles de tâches complexes.

La *première section* de ce chapitre présente une analyse des activités des opérateurs dans une salle ou une passerelle de contrôle. Cette analyse s'appuie sur un état de l'art et sur des projets réels de conception des systèmes de contrôle-commande. Elle permet de mettre en évidence la complexité dans la conception des modèles de tâches de supervision.

La *deuxième section* de ce chapitre dresse un état de l'art non exhaustif décrivant quatre des approches existantes qui proposent des solutions pour résoudre ce problème de complexité.

La présentation de ces approches est ensuite suivie d'une analyse comparative qui fait l'objet de la *troisième section* de ce chapitre.

La *quatrième section* de ce chapitre pose la problématique qui émerge de l'état l'art réalisé. Nous y mentionnons les limites de l'état de l'art relatives au passage du pattern aux modèles de tâches vérifiés et validés.

Suite à cela, nous proposons une démarche présentée dans la *cinquième section* de ce chapitre dont le but est de répondre à la problématique.

1. Analyse de la tâche humaine dans la conception des systèmes de contrôle commande

La description en modèles de tâches des activités de l'utilisateur sur un système de contrôle-commande se base sur la formalisation des spécifications. Cette formalisation est fondée d'une part sur une revue et analyse de la littérature (AFNOR X50-151; Bovell, Carter, et Beck 1998; Kluge 2014) ainsi que, d'autre part, sur un retour d'expérience industriel acquis par l'analyse de spécifications établies dans le cadre de projets réels et par des entretiens menés auprès d'experts en conception. Cette formalisation permet d'**identifier** les informations nécessaires contenues dans la spécification et qui doivent être modélisées en tâches. La modélisation en tâches de ces informations fait ressortir des actions récursives globales, nécessaires au pilotage d'un système de contrôle-commande.

La supervision des systèmes contrôle-commande consiste à réaliser un certain nombre de commandes en fonction du contexte (l'état du système, les besoins des utilisateurs, etc.) et à surveiller les affichages à travers les IHM. Ces tâches sont qualifiées de « *complexes* » et classées dans quatre catégories par (Bovell, Carter, et Beck 1998).

La première catégorie concerne la ***surveillance de fonctions du système***. Cette tâche est la principale car elle occupe en grande partie le temps des opérateurs. En effet, c'est au cours de cette tâche que les opérateurs détectent les problèmes et prennent des décisions pour les résoudre. Elle consiste en la *surveillance des niveaux d'alarmes des composants* et *du système*, des *variables du procédé*, de l'*état du système* pour détecter les anomalies.

La deuxième catégorie concerne la ***manipulation des commandes*** qui est la deuxième tâche importante pour l'opérateur. Cette tâche peut être réalisée à tout moment (lorsque le système est en régime permanent, lorsque des défauts sont détectés sur le système, etc.). Dans un contexte fonctionnel, pour la plupart des systèmes de contrôle-commande, les manipulations de commande se produisent soit au démarrage d'une fonction, soit lorsque la fonction s'arrête normalement, soit lorsque des défauts sont renvoyés (Bovell, Carter, et Beck 1998). Quatre actions possibles sont inclues dans la manipulation des commandes : *Manipuler les commandes prévues*, *Effectuer des actions réparatrices et correctives*, *Effectuer des actions réparatrices immédiates*, *Restaurer les fonctions du système*.

- En ce qui concerne les *manipulations de commandes prévues*, elles correspondent principalement au lancement des fonctions de haut-niveau du système. Elles sont effectuées par procédure écrite, pour répondre aux besoins des opérateurs et du système. Pour veiller à ce que les étapes de la procédure exécutée en séquence soient effectuées, une confirmation est requise après réalisation de chaque étape.
- Les *actions réparatrices et correctives* sont effectuées par l'opérateur, lorsqu'il détecte des défauts mineurs sur le système. Les défauts mineurs sont des défauts prévus lors de la conception du système, dont l'apparition n'a pas d'impact considérable sur le fonctionnement du système. L'opérateur doit tout de même effectuer des actions correctives, en respectant les spécifications techniques, pour empêcher que le système ne se dégrade durablement.
- Les *actions réparatrices immédiates* sont effectuées immédiatement après un accident (Définition 1). Les opérateurs sont tenus de mémoriser ces actions afin de pouvoir les exécuter sans s'appuyer sur des documents contenant des procédures. Ces actions sont référencées dès que possible dans les documents, pour s'assurer qu'elles ont toutes été effectuées. Elles sont effectuées indépendamment de la cause du défaut, et consistent à retirer du système l'élément qui ne fonctionne plus, et à activer en remplacement d'autres éléments nécessaires au bon fonctionnement du système.
- La *restauration des fonctions du système* est nécessaire après de graves accidents, et est effectuée en suivant des procédures écrites. Ces procédures ont été suivies par les opérateurs, uniquement pendant la formation sur un simulateur. Cependant, la réponse du système réel peut-être différente et inconnue. Par conséquent, Les opérateurs doivent faire preuve d'une extrême prudence. Comme pour la plupart des autres types de fonctionnement, l'opérateur active une commande et attend la réaction appropriée de la part du système. La réaction attendue devrait indiquer que l'action a eu l'effet désiré. Bien que les actions dans de tels cas doivent être effectuées rapidement, les opérateurs doivent vérifier que leurs actions permettent d'avoir les résultats souhaités, sinon ils doivent effectuer une autre action. Ce processus est répété jusqu'à ce que le système soit dans un état sûr et stable.

La troisième catégorie concerne le ***diagnostic de défauts***. Cette catégorie de tâches intervient lorsqu'un défaut est identifié par l'opérateur en supervision. Il s'agit alors de diagnostiquer

correctement le ou les défauts et de prendre les mesures appropriées pour les corriger. Pour réaliser la tâche de diagnostic de défaut, il faut tout d'abord « *Analyser le problème* », puis « *Sélectionner les meilleures alternatives* ». Généralement les opérateurs sont formés et connaissent les actions à réaliser et les décisions à prendre face aux différentes situations.

(Bovell, Carter, et Beck 1998) définit un **accident** sur un système de contrôle-commande comme un évènement qui impliquent de multiples défauts sur les équipements et/ou des erreurs faites par les opérateurs.

Définition 1 un accident au sens de la supervision

La quatrième catégorie concerne les ***tâches administratives***. Elles regroupent la réalisation des « *tests de surveillance* » et la « *vérification des spécifications techniques* » qui sont des tâches qui sont effectuées pour assurer la sécurité les installations en cas d'accident. On y retrouve également la « *tenue du journal* » et les « *rapports d'évènement* ».

Ces quatre catégories de tâches sont étroitement liées dans une salle ou une passerelle de contrôle. La tâche de *surveillance de fonction* est réalisée en permanence par un opérateur pendant que les autres s'occupent des tâches administratives. La tâche de « *supervision* » devient beaucoup plus difficile lorsque le système est dans un état anormal. En effet, lorsque l'opérateur détecte un défaut sur le système (une alarme ou un élément en défaut) pendant la supervision, il doit analyser et faire l'action appropriée pour corriger le défaut. Les opérateurs doivent manipuler les commandes pour remettre le système en fonctionnement normal. Ensuite, l'opérateur continue la tâche de supervision.

Bovell, Carter et Beck ont présenté un flot qui montre la liaison entre les quatre catégories d'actions à effectuer par les opérateurs en supervision (Bovell, Carter, et Beck 1998). Ce flot ne permet cependant pas de prendre en compte avec précision la hiérarchisation des activités de l'opérateur.

Kluge a présenté une table contenant les séquences d'actions à réaliser par un opérateur en supervision (Kluge 2014). La réalisation de cette table s'appuie sur un état de l'art et sur des entretiens effectués dans l'industrie. Cette table regroupe les activités des opérateurs en sous-objectifs définis en termes de supervision, prise de décision, communication, actions, etc. En effet, lors de la supervision du système, les actions à réaliser sont généralement groupées en fonction du contexte. Cependant, cette table ne permet pas de structurer ces actions avec la précision nécessaire afin de bien mener les missions complexes (Martinie 2011) et variées des opérateurs.

Les commandes prévues sur les systèmes de contrôle-commande correspondent aux commandes de haut-niveau permettant à l'opérateur de lancer les fonctions de haut-niveau et aux commandes nécessaires à la supervision (par exemple, la navigation entre les sous-systèmes de l'installation supervisée).

Regrouper toutes les informations issues de la formalisation en termes de modèles de tâches nous semble être une meilleure solution pour une bonne description de l'activité des opérateurs surveillant et contrôlant un nombre important de paramètres. En effet, les tâches des opérateurs en supervision sont considérées comme des tâches complexes (Martinie 2011) or pour exécuter une tâche complexe, la mise en place d'une séquence d'actions est nécessaire (Bainbridge et Dorneich 1999).

Généralement, les modèles de tâches de supervision tiennent compte des spécificités de chaque fonction de haut-niveau du système. On imagine bien la complexité dans la conception de tels modèles.

2. La conception des modèles de tâches complexes

Plusieurs travaux ont tenté d'apporter une solution à la complexité de la conception des modèles de tâches. Ainsi, pendant le *workshop SIGCHI'97* sur les patterns (Bayle et al. 1998), les participants ont considéré les patterns comme un moyen pour résoudre les problèmes de complexité et de diversité qui augmentent dans la conception des IHM. L'objectif principal des patterns est de créer un inventaire de solutions pour aider les concepteurs d'interface à résoudre les problèmes fréquents et difficiles de conception. Les patterns ont été introduits pour la première fois dans le domaine de l'architecture par Christopher Alexander (Alexander et al. 1977; Alexander 1979). Ils ont ensuite été introduits dans le domaine du génie logiciel à travers les travaux de (Gamma et al. 1995). Un pattern est défini comme un format de description de solutions pour les problèmes récurrents de conception (Granlund, Lafrenière, et Carr 2001).

L'un des premiers travaux sur l'intégration des patterns dans la modélisation des tâches est celui de (Breedvelt-schouten, M., Paternò, et Severijns 1997) qui étaient motivés par la possibilité de réutiliser de bonnes solutions de conception pour résoudre les problèmes récurrents liés à la spécification du dialogue, afin de diminuer le temps de conception. En effet, après plusieurs conceptions de modèles de tâches, ils se sont rendus compte de la possibilité d'identifier des patterns de tâches spécifiques dans la structure arborescente décrivant les modèles de tâches. Ces patterns peuvent alors être réutilisés dans la conception de différentes applications présentant des exigences communes.

L'objectif de leurs travaux est de construire une structure réutilisable en modèle de tâches pour permettre aux concepteurs de se concentrer sur les besoins de l'utilisateur. Ce faisant, ils facilitent la conception des applications. L'utilisation des patterns ayant pour but de permettre aux concepteurs de créer facilement des modèles à partir des travaux d'autres concepteurs, ils pensent donc que si les patterns existaient pour les modèles de tâches, cela améliorerait la conception de ces modèles, facilitant ainsi la réutilisation. Leurs travaux ont permis de mettre en évidence l'utilisation des structures de tâches pour accélérer le processus de construction de modèles de tâches et leur intégration dans la conception des applications industrielles de grandes envergures.

D'autres travaux se sont également intéressés à l'intégration des patterns dans le processus de conception.

2.1. Le framework Pattern-Supported Approach (PSA)

Granlund, Lafrenière et carr présentent dans leurs travaux, une approche basée sur les patterns (Pattern Supported Approach (PSA)) destinée à la conception des interfaces utilisateurs dans le contexte de la visualisation d'information (Granlund, Lafrenière, et Carr 2001). PSA exploite les patterns avant et pendant le processus de conception (Figure 22). En effet, les auteurs ont décrit les patterns pour les différents modèles de conception. Cette approche offre une double chaine de liaisons entre les patterns de chaque étape du processus de développement (Figure 23). Ces liaisons permettent de capturer la connaissance des modèles précédents et de les communiquer au modèle suivant. Dans le Framework PSA, le

pattern de tâches est utilisé pour capturer et transmettre la connaissance sur la tâche (les utilisateurs et le contexte à partir de projets similaires précédents). Il est également utilisé pour suggérer une solution appropriée pour la conception interactive. Le PSA offre un point d'entrée à un langage mature de pattern correspondant à la description des classes et suggère (sans limitation d'utilisation de patterns) une chaine de patterns adaptée à différents niveaux d'analyse et de conception.

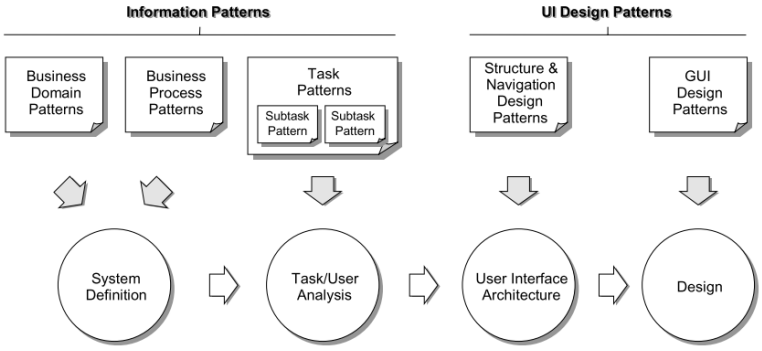


Figure 22 le Framework PSA

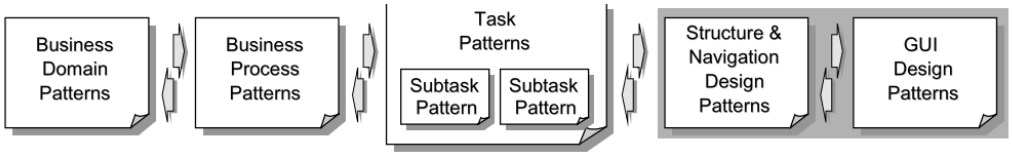


Figure 23 Liaison entre les patterns du PSA

Avec un exemple concret du domaine des télécommunications, les auteurs se focalisent sur les patterns de tâches et de sous-tâches pour illustrer la connaissance d'une tâche et les interactions appropriées que la solution de conception peut capter et communiquer.

Dans PSA, nous allons nous focaliser sur les patterns de tâches. Ces patterns décrivent des tâches complexes. Ils contiennent des patterns de sous-tâches qui décrivent à leur tour des parties de la tâche. La description des patterns de tâches contient les informations suivantes : le nom, le contexte, le problème (de conception), l'exemple (pour clarifier la tâche), les forces (les facteurs conflictuels qui influencent la conception directement ou indirectement), la solution de conception, les patterns de sous-tâches, les patterns de structure et de navigation résultants, les patterns de conception de l'interface graphique résultants.

Les patterns de sous-tâches sont dérivés du principe que toutes les tâches complexes sont décomposables en petites tâches : les sous-tâches. En utilisant les langages de patterns, on peut mettre des patterns de sous-tâches dans des patterns de tâches. Les patterns de sous-tâches sont génériques et indépendants de la tâche utilisateur et du contexte. Dans PSA ils sont composés des mêmes sections que les patterns de tâches.

2.2. L'outil Task Pattern Wizard

Gaffar et ses collègues apportent une réponse à l'une des questions posées par (Trætteberg 2002) qui est : « Comment les modèles peuvent-ils être utilisés comme une représentation du système physique dans le développement des systèmes interactifs ? » (Gaffar et al. 2004).

La motivation derrière ces travaux est la réutilisabilité des modèles. En effet, l'activité de conception des systèmes de contrôle-commande reste difficile. Les modèles sont proposés pour faciliter la conception. Cependant, la création de différents modèles et le processus de

liaisons entre les modèles est fastidieux. La plupart des modèles manquent de flexibilité, de réutilisabilité (MOBI-D9, TADEUS10, TERESA11). Seuls quelques-uns offrent la possibilité d'une réutilisabilité sous forme de copier/coller. Or cette forme de réutilisabilité n'est pas adaptée au cycle de vie de développement des systèmes interactifs. En effet, avec la réutilisation sous forme de Copier-coller, le concepteur qui réutilise le pattern, copie un modèle de composant et le change avec de nouvelles exigences sans maintenir une cohérence avec le composant d'origine (Mens, Lucas, et Steyaert 1998).

L'utilisation des patterns pour compléter l'approche basée sur les modèles, peut être une bonne solution pour faciliter la construction et les transformations de modèle et donc leur réutilisabilité. L'introduction des patterns dans la conception a été inspirée par les travaux de (Breedvelt-schouten, M., Paternò, et Severijns 1997). Contrairement à ces travaux, ceux de (Gaffar et al. 2004) n'utilisent pas l'approche orientée pattern seulement pour la conception des blocs de modèles de tâches mais également pour la conception des modèles plus concrets dans le cycle de développement des systèmes interactifs.

Pour manipuler plus facilement les patterns, ils ont proposé l'outil Task Pattern Wizard qui permet de sélectionner, d'adapter et d'intégrer les patterns pour la création de modèles de tâches. Cet outil est basé sur un langage de patterns : Task Pattern Markup Language (TPML) (Sinnig 2004). La structure de base du TPML (Figure 24) est composée de 5 éléments descripteurs : Nom, Problème, Contexte, Solution et Logique. Les éléments descripteurs sont principalement utilisés pour sélectionner un pattern jugé approprié par l'utilisateur. Contrairement à la description du pattern, la structure du pattern (pattern body) capture l'essentiel des informations formalisées du pattern de tâches (tâches, sous-tâches, ordre relation etc.).

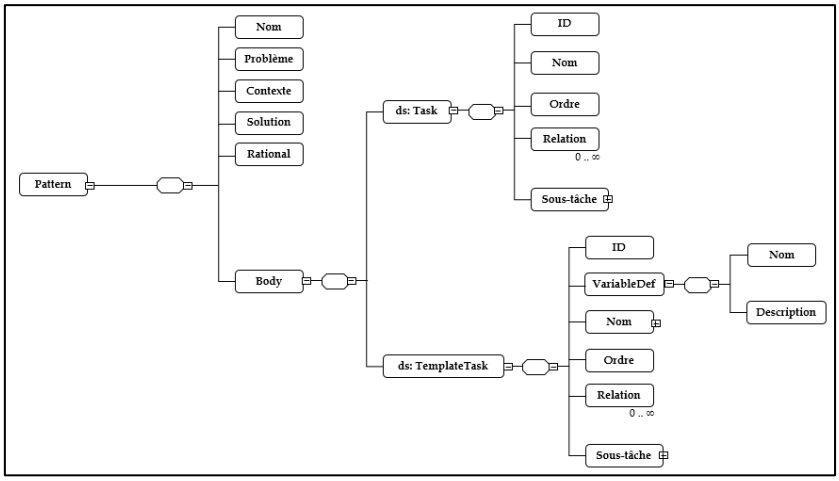


Figure 24 Structure de TPML.

L'outil Task Pattern Wizard est capable de lire et de visualiser des modèles de tâches existants qui sont décrits en XIML[12]. Un schéma TPML XML est proposé pour la description des patterns de tâches. L'outil est capable d'interpréter les descriptions des patterns de tâches faites avec le langage TPML.

[9] http://smi-web.stanford.edu/projects/mecano/mobi-d.htm
[10] http://xml.coverpages.org/MuellerDSVIS2001.html
[11] http://giove.cnuce.cnr.it/teresa.html
[12] http://www.ximl.org/

Après l'analyse du pattern, l'outil guide l'utilisateur, étape par étape dans le processus d'intégration et d'adaptation. Une fois que le pattern a été adapté au contexte d'utilisation, l'outil insère le fragment de tâches résultant dans le modèle de tâches. Les patterns sont ici des fragments réutilisables.

2.3. L'outil Pattern In Modelling (PIM)

Radeke et ses collègues présentent dans leurs travaux, un outil permettant la réutilisabilité des patterns et leur transformation en modèles de tâches à partir des patterns prédéfinis (F Radeke et al. 2006). La Figure 25 présente l'interface de l'outil. Sur cette Figure 25 la Zone 1 présente les différents types de patterns (patterns de modèle de tâches, de dialogue, de présentation et d'architecture). La zone 2 présente le modèle de tâches à manipuler. La zone 3 présente les patterns prédéfinis. La conception d'un modèle de tâches revient à identifier, à sélectionner, à instancier et à intégrer un ou plusieurs de ces patterns prédéfinis.

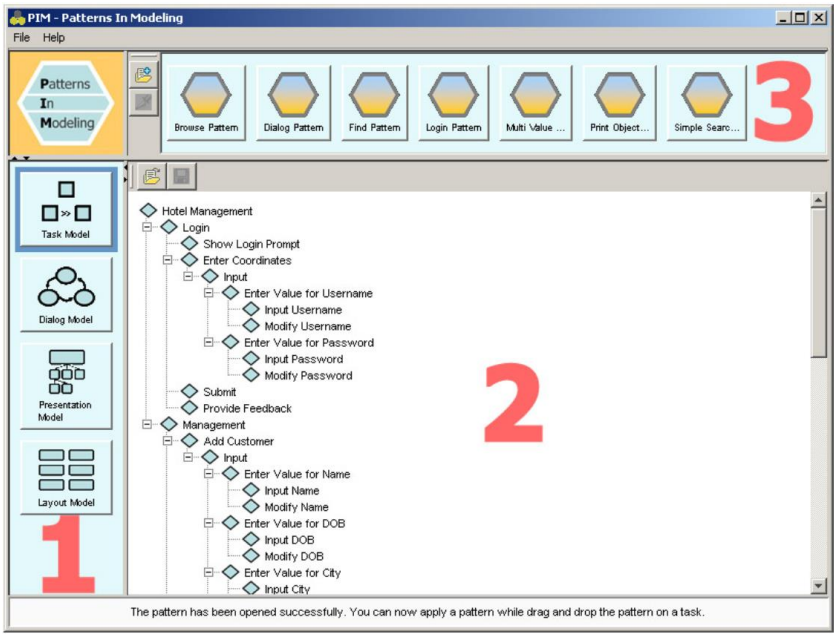


Figure 25 L'interface utilisateur de l'outil PIM

Le processus d'utilisation des outils proposé se fait en quatre phases : *Identification*, *Sélection*, *Instanciation* et *Intégration*. La première phase consiste à identifier dans un modèle de tâches, les patterns utilisés. La Figure 26 (a) présente un modèle de tâches décrit sur CTTE et comprend les tâches *Login In* et *Select station* qui sont des patterns de tâches dans PIM. L'outil contient une transformation de modèle qui permet de convertir les fichiers de description de tâches de CTT en XIML, le seul langage de spécification accepté par PIM.

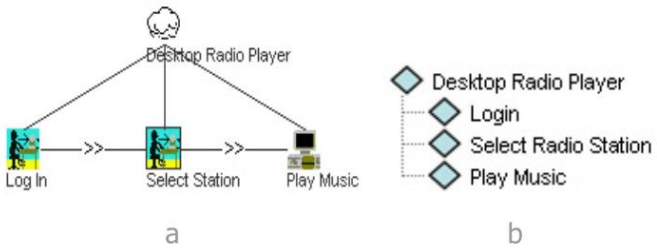


Figure 26 Modèle de tâches pour écouter de la musique à travers une station radio. (a) décrit avec CTTE. (b) une fois chargée sur PIM.

La Figure 26 (b) montre la structure du modèle de tâches une fois chargée dans PIM. Donc uniquement les modèles de tâches définis sous CTT peuvent être utilisés par cet outil. En plus, après l'identification des patterns que contient le modèle de tâches, le concepteur peut choisir d'insérer le pattern, soit en tant que tâche optionnelle, soit en tant que tâche itérative.

La phase de *sélection* se base sur les patterns de tâches proposés par l'outil. Il suffira de faire un glisser/déposer des patterns de tâches pour les intégrer dans les modèles de tâches. Par exemple, dans le modèle de tâches de la Figure 26 (b), ils font un glisser/déposer du pattern « Login pattern » présent sur la Figure 25 (zone 3). Une fois le pattern intégré dans le modèle de tâches, il faudra l'instancier car les patterns sont génériques. La phase d'instanciation se fait en deux étapes : l'instanciation de la structure et la définition des valeurs des variables contenues dans les patterns. L'instanciation de la structure concerne la sélection des parties optionnelles des patterns. En effet, certains patterns contiennent des parties optionnelles et lors de l'instanciation le concepteur peut choisir de les garder ou non. La Figure 27 montre le pattern Login et sa description qui permet au concepteur de l'instancier.

Pendant la phase d'intégration, l'instance de la phase précédente est intégrée dans le modèle cible. En effet, l'outil PIM convertit l'instance au format du modèle et l'intègre dans le modèle cible en remplaçant les parties sélectionnées pendant la première phase.

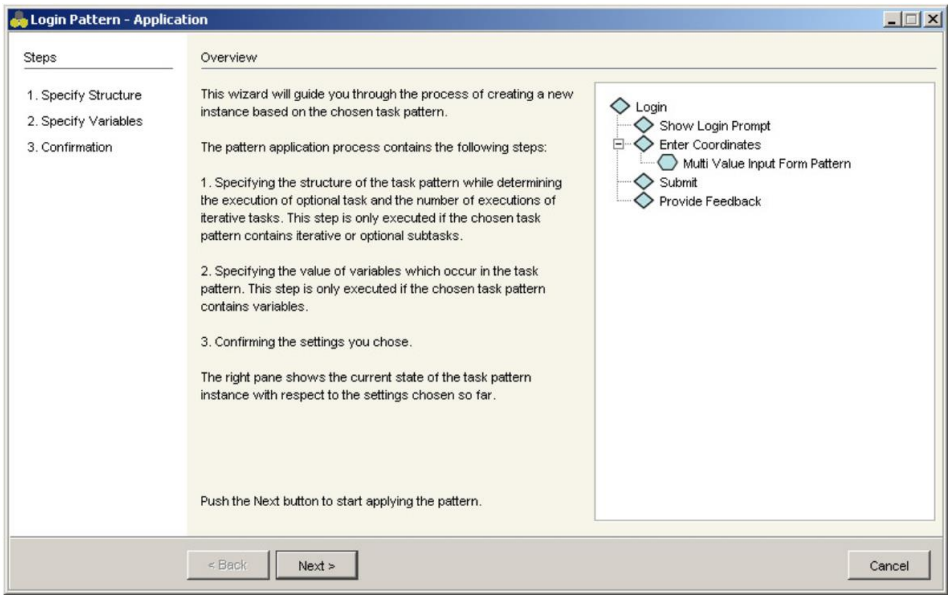


Figure 27 Pattern Login et sa description

Ces travaux ont permis de démontrer qu'à travers un outil on peut passer des patterns de tâches aux modèles de tâches. L'outil présenté permet de combler le fossé entre les approches basées sur les modèles et les approches basées sur les patterns (Figure 28).

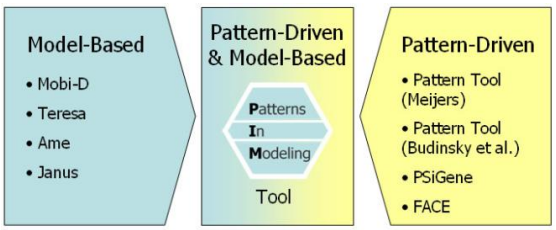


Figure 28 Objectif de l'outil PIM

Cette méthode pourra être appliquée à tout modèle de descriptions d'interface quel que soit son niveau : modèle de dialogue, de présentation, etc. Cependant, ce Framework a besoin d'être formalisé (F Radeke et al. 2006). Cette formalisation implique l'analyse des notations formelles pour la spécification des modèles d'interfaces utilisateur ainsi que les patterns correspondants.

2.4. Le Framework Pattern-Driven and MBUI (PD-MBUI)

La motivation derrière ces travaux vient des limites des outils basés sur les modèles proposés et du fait qu'ils sont rarement utilisés. Les outils proposés pour mettre en œuvre les approches basées sur les modèles fournissent un support marginal qui n'est pas acceptable dans plusieurs domaines de l'industrie car ils ont des ressources limitées, de la concurrence et peu de temps. Ces défauts peuvent être attribués au fait que les méthodes basées sur les modèles telles que MOBI-D et TERESA manquent de flexibilité dans la réutilisation, la conception et la transformation des modèles. Au mieux, seulement quelques approches offrent une forme de copier-coller pour la réutilisation. De plus, dans ces cas ce sont les utilisateurs qui font simplement une copie du modèle de composant et le changent manuellement en fonction des nouvelles exigences. Avec le copier-coller aucune cohérence avec la solution originale n'est maintenue (Mens, Lucas, et Steyaert 1998). L'analyse du copier-coller et le concept de modèles fragmentés sont insuffisants lorsque l'on tente d'intégrer la réutilisation d'une manière systématique et rétractable dans le cycle de développement des interfaces utilisateurs basées sur les modèles. Les patterns sont utilisés comme solution de réutilisabilité. Afin de proposer un outil de capture de connaissance pour les approches basées sur les modèles, deux points essentiels sont abordés dans leurs travaux :

- Une classification des patterns en fonction des modèles doit être faite. Une telle classification doit différencier les patterns qui sont des blocs de conception pour les modèles, des patterns qui dirigent la transformation des modèles et des patterns qui créent l'interface concrète.
- Un outil qui peut assister les développeurs dans la sélection de leurs propres patterns, l'instanciation et la création d'un modèle.

Pour répondre aux problèmes de réutilisation des différents modèles et tenir compte de tous les aspects de conception, l'approche décrite sur la Figure 29 a été proposée. L'objectif de cette approche est d'unifier dans un seul Framework les approches basées sur les patterns et sur les modèles, deux puissantes méthodes pour l'ingénierie logicielle et celle des interfaces utilisateur. Un outil Patterns wizard aide le développeur de l'interface utilisateur dans la sélection et l'application des patterns lors de la construction et la transformation des divers modèles pour une interface concrète. Cet outil passe dans l'arbre de patterns de tâches et signale à l'utilisateur lorsqu'il rencontre une variable non résolue. Ce Framework intègre les notations XUL[13], UML[14] et CTT (F. Paternò, Mancini, et Meniconi 1997). Les auteurs se sont assurés que la librairie de patterns proposée est valide car ils ont inclus les patterns proposés par les travaux de (Gaffar et al. 2004; F. Paternò 2012; Sinnig 2004). Les composants ou widgets de l'interface utilisateur tels que les boutons, les fenêtres et les boites de dialogues contenus dans l'outil sont des objets de Java Swing.

[13] http://xulfr.org/
[14] http://www.uml.org/

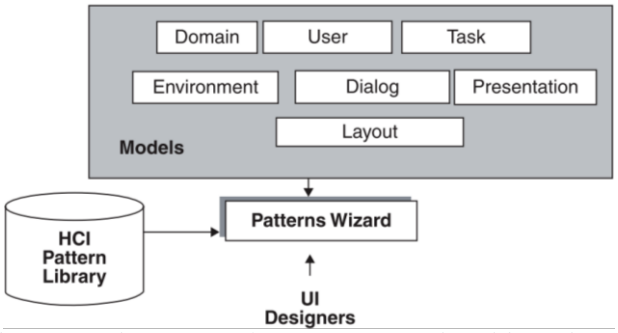


Figure 29 le Framework PD-MBUI (Pattern Driven and Model Based User Interface)

La bibliothèque de patterns contient les patterns de tâches, les patterns de dialogues, les patterns de présentation, les patterns d'architecture.

- Les patterns de tâches sont utilisés pour décrire la structure hiérarchique des fragments de tâches. Ces fragments peuvent être utilisés comme des blocs de conception de tâches pour construire graduellement le modèle de tâches souhaité.
- Les patterns de dialogue sont utilisés pour regrouper les tâches et pour suggérer des séquences entre les vues de dialogue.
- Les patterns de présentation sont utilisés pour faire correspondre les tâches complexes à un ensemble d'éléments d'interaction prédéfinis qui sont identifiés dans le modèle de présentation.
- Les patterns d'architecture sont utilisés pour établir certains styles qui sont ensuite capturés par le modèle d'architecture.

Dans les perspectives énoncées par les travaux de (F Radeke et al. 2006), il était question de compléter l'approche afin que l'outil proposé puisse non seulement permettre d'instancier et d'adapter les patterns de tâches, mais également les patterns de tous les modèles intervenant dans le processus de conception d'interface à base de modèles.

Nous nous concentrons sur les patterns de tâches. Les patterns de tâches décrivent des fragments de tâches génériques et réutilisables qui servent à définir le modèle de tâches. Une fois que tous les patterns ont été adaptés, le modèle de tâches résultant peut être simulé avec XUL-Task-Simulator (Sinnig 2004). En particulier, les exemples de patterns de tâches peuvent être utilisés comme des blocs de conception pour les modèles de tâches. Les exemples de tels patterns sont : « *Find something* », « *Search something* », « *Buy something* », « *Search for something* » ou « *Login to the system* ».

2.5. Le Framework pour Smart Environment

En voulant appliquer les techniques décrites dans (Zaki et Forbrig 2012) à un cas d'application, un « smart environment », les auteurs se sont rendus compte que le langage de patterns n'est pas adapté pour décrire les tâches de cet environnement.

Les « Smart environment » sont définis comme « un petit monde dans lequel différents types d'appareils intelligents travaillent continuellement pour faire vivre plus confortablement les habitants » (Cook et Das 2004). Une bonne compréhension des tâches que l'utilisateur veut accomplir est une étape obligatoire pour faire réagir la salle de façon homogène et appropriée. Par conséquent, la modélisation du comportement de l'utilisateur dans l'environnement est recommandée. Cela permet d'avoir une assistance optimale et réussie des éléments de l'environnement. Pour bien modéliser les tâches à exécuter dans cet environnement, il faut

prendre en compte les conditions et les paramètres environnementaux qui interviennent et affectent la façon dont les tâches sont réalisées. Un modèle de tâches simple isolé de contraintes environnementales ne permet donc pas d'exprimer la façon exacte dont les tâches doivent être exécutées dans l'environnement. Pour une bonne expression des scénarios dans ce domaine, (Wurdel, Sinnig, et Forbrig 2008) ont développé le langage CTML (Collaborative Task Modelling Language). Le CTML semble être une bonne solution pour la description des scénarios ayant lieu dans ces types d'environnement. Cependant, la conception des différents modèles (modèles de tâches, modèle de dispositifs, modèle de domaine, etc.) est fastidieuse pour les développeurs. Les dépendances réciproques entre toutes les entités dans un tel environnement augmentent la complexité du processus de modélisation, les erreurs et le temps de conception.

Pour résoudre ces problèmes, les auteurs ont proposé la définition d'un langage de patterns adapté au domaine de Smart Environment. Toutefois, les auteurs précisent que la méthodologie utilisée pour définir leurs patterns peut également être utilisée pour définir des patterns similaires pour d'autres domaines. La démarche proposée ici commence par la détermination de différentes catégories de patterns pour un smart meeting room Figure 30. Ensuite, pour proposer des patterns réutilisables, ils ont identifié tous les objectifs récurrents des équipes. Cette identification s'est basée sur des exemples concrets de scénarios dans un Smart Environment. Suite à cela, *Goal-based pattern* a été créé pour fournir un modèle global pour tous les objectifs des équipes. De plus, en analysant ce pattern, ils ont fait ressortir un ensemble d'entités de patterns (*Figure 30*). Ils ont développé un ensemble de patterns pour tous les aspects environnementaux considérés. Le diagramme d'état-transition a été utilisé pour décrire *device-based Template.* La notation UML a été utilisée pour décrire les *Domain-based patterns*. Les modèles de tâches ont été utilisés pour construire le *Team-based pattern*. *Le Role-based pattern* a été défini avec CTML. Ce pattern est une forme abstraite des patterns de tâches et définit un scénario répétitif complet pour l'environnement. Le *Role-based patterns* peut lui-même être décomposé en patterns de tâches.

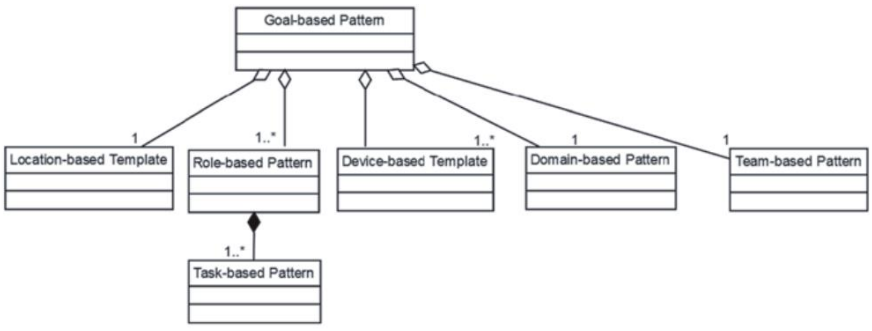


Figure 30 Les catégories de patterns pour un smart meeting room

Le Tableau 1 représente l'un des patterns de tâches, le *Present Slide pattern*. La structure des patterns a été représentée en CTT. Dans la partie diagramme, on remarque que la solution est divisée en trois différentes parties. Tout d'abord, le fragment de tâches qui doit être chargé par le concepteur et intégré dans son modèle de tâches est fourni.

Ce fragment décrit les tâches qui doivent être réalisées par les acteurs qui se trouvent dans la salle. Ensuite, un diagramme UML est utilisé pour identifier les entités pertinentes de toutes les tâches représentées dans le fragment de tâches. Enfin, une représentation formelle de toutes les contraintes d'exécution des tâches individuelles est définie par un diagramme d'activité

UML. Le diagramme d'activité fournit les préconditions et les post-conditions pour permettre de visualiser toutes les contraintes liées à l'exécution des tâches. Chaque partie de la solution peut être adaptée aux différents contextes d'utilisation de ce domaine.

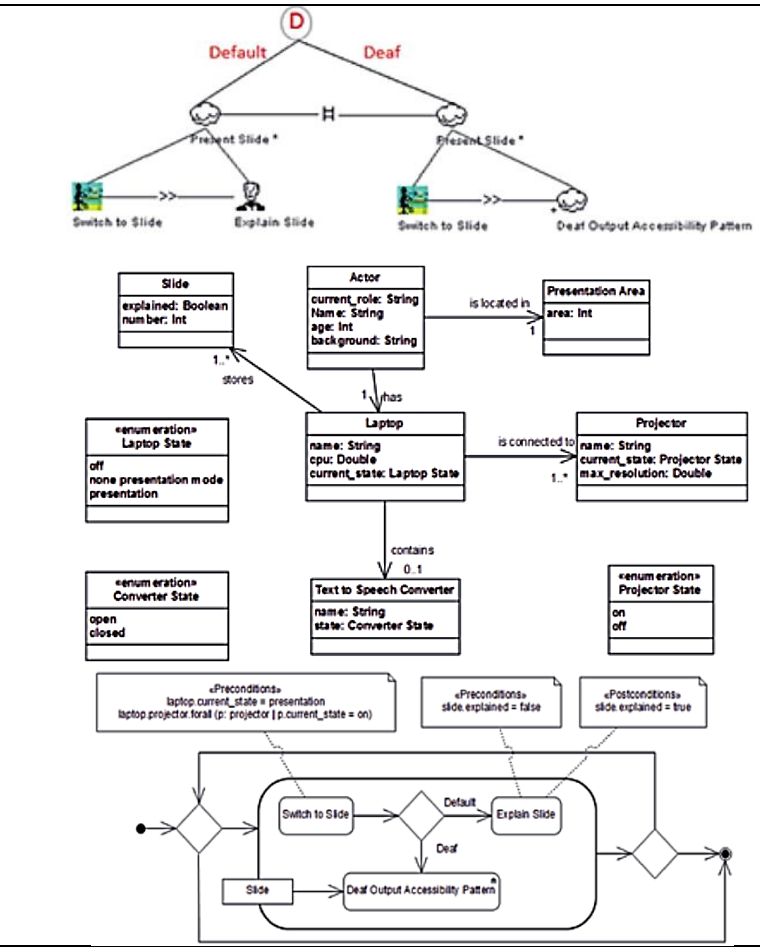
ID	1
Name	Present Slides
Problem	Use the projector and the presenter device in order to present some slides to the audience.
Situation	A given user in the environment has to present some slides on the canvas. This may be needed in a conference session, lecture or a discussion.
Solution	The actor who is performing this task needs to iterate over all the slides of his/her presentation and to explain them by one. As a pre-request, he/she should be located in the presentation area, having the slides to be presented stored on his/her presenter device which is connected to the projector in use. The number of projectors needed depend on the presentation mode (e.g: the smart room gives the user opportunity to use only one projector for his presentation, or alternatively several ones in case the slides should be presented on more than one canvas). Only in case the presenter is deaf, he/she can use a text to speech converter to present the slides to the audience.
Diagram I. Task hierarchy II. Environmental dependencies III. Execution constraints visualization	
Adaptation variables	Number of projectors, kind of user impairment
Referenced patterns	Deaf Output Accessibility

Tableau 1 Pattern de tâches pour « present slides » (Seffah, 2015)

Pour utiliser efficacement le langage de pattern, le concepteur doit commencer par vérifier si le scénario de la tâche qu'il est sur le point de développer correspond à l'un des patterns *goal-based patterns* existant dans le langage. Si c'est le cas, le pattern est chargé et le concepteur affecte les valeurs concrètes pour adapter le pattern. Malgré le fait que les auteurs aient essayé

d'identifier tous les objectifs en équipe (*team-based goal*) pour lesquels il est possible qu'un groupe d'acteurs se regroupe dans un smart meeting room, il est toujours possible d'avoir des scénarios qui ne sont pas représentés par des patterns. Dans ce cas, le concepteur peut être assisté par les *entity-based patterns*. Ainsi, il/elle peut charger un modèle complet pour une ou plusieurs des entités incluses (par exemple, un modèle de dispositif pour l'un des dispositifs existants). Cependant, il est évident que l'aide fournie par ces patterns est inférieure à celle fournie par les précédents, car le concepteur doit encore construire les modèles non chargés; et en plus, tous les modèles doivent être liés au modèle de tâches. Le concepteur peut également tirer parti des *role-based patterns* existants. Dans le cas où aucun des rôles complets correspondants ne sont utilisables, le concepteur peut encore tirer profit des patterns de tâches qui fournissent des fragments de tâches répétitives, qui peuvent être intégrés dans son modèle de tâches. Enfin, ces modèles de tâches offrent certaines informations dont le concepteur peut avoir besoin pour construire les autres modèles liés à l'environnement.

3. Analyse comparative des approches présentées

Dans la section précédente, nous avons présenté quelques approches qui utilisent les patterns de tâches pour résoudre les problèmes de conception des modèles de tâches. Nous proposons dans cette section une analyse comparative des outils et approches existants afin de mettre en évidence leurs avantages et inconvénients.

Le Framework PSA, utilise les patterns pendant tout le cycle de développement du logiciel, afin d'étendre la réutilisabilité au niveau de tous les modèles de conception. Ce Framework offre une bonne solution de réutilisabilité et permet de capturer et de propager les informations entre les patterns proposés (les patterns de tâches, de domaine, de structure et navigation, etc.). Cependant, le concept de liaison des patterns dans une conception n'est pas abordé (Seffah 2015). De plus, l'approche ne traite pas le passage des patterns de tâches aux modèles de tâches.

L'approche de Gaffar et ses collègues a insisté sur un autre aspect important de la notion de pattern : la combinaison de patterns (Gaffar et al. 2004). En combinant différents patterns, les développeurs peuvent utiliser les relations de patterns et les combiner afin de produire une solution de conception efficace. Cette approche offre une démarche et un outil permettant de faciliter l'intégration et l'adaptation des patterns en modèles de tâches, puis le passage des modèles de tâches aux interfaces concrètes. Dans leurs travaux, plusieurs patterns ont été proposés pour représenter des modèles de tâches génériques et des techniques pour les transformer en une interface concrète. Cependant, cet outil utilise en plus des patterns de tâches, les patterns de dialogues et lorsque l'utilisateur veut atteindre un simple objectif, il doit faire plusieurs actions et prendre plusieurs décisions de façon consécutive avant d'atteindre cet objectif. La démarche de manipulation des patterns est donc lourde (Seffah 2015). De plus, pour être interprété par l'outil, les modèles de tâches doivent être exprimés uniquement en XIML. Pour utiliser les modèles de tâches décrits en CTT, les auteurs utilisent Dialog graph editor (A. Wolff et al. 2005), pour convertir les fichiers de CTT en fichiers XIML.

Le Framework PD-MBUI propose une approche très intéressante permettant d'instancier et d'adapter tous les modèles intervenant dans le processus de conception, et pas seulement les modèles de tâches. Cependant elle se base sur une bibliothèque de patterns formalisés et prédéfinis qui ne sont pas utilisables dans tous les domaines de la conception. Actuellement les auteurs prévoient d'étendre le concept de modélisation dans un environnement de patterns

intégré qui contiendra leur outil et d'autres outils basés sur les patterns généralisés qui seront indépendants de toute plateforme et langage de programmation.

Dans (Seffah 2015), un langage de pattern a été construit pour le domaine de Smart meeting room. Ce langage peut assister le concepteur dans la construction des modèles et rendre ainsi le processus meilleur et moins couteux en temps. Les patterns de tâches appartenant à ce langage permettent une intégration des fragments de tâches comme bloc de conception à l'intérieur du modèle de tâches de l'utilisateur. L'ensemble de la méthodologie est adaptable à la conception des patterns similaires dans d'autres domaines. De plus, la prise en compte des pré et post conditions dans la définition des contraintes de visualisation liées aux patterns de tâches apporte une précision dans la description de ces derniers. Cependant, le manque d'outil ne facilite pas l'utilisation de ce langage.

Pour construire les patterns certains auteurs ont utilisé la notation CTT; d'autres des langages comme UsiPXML (Frank Radeke et Forbrig 2007) structuré suivant le format PLML ; et d'autres simplement le TPML. Le TPML était issu du workshop de *CHI 2003 (Conference on Human Factors in Computing Systems)* avec l'objectif de permettre la définition d'une structure commune pour les patterns de tâches. Avec ce langage, l'utilisation des patterns pour la définition des modèles de tâches nécessite des plateformes qui peuvent interpréter le langage.

Par ailleurs, en plus des approches de patterns de tâches présentées, HAMSTERS (Martinie 2011) fournit un support à la modularité et à la réutilisabilité des modèles de tâches, à travers les éléments de notation *subroutine* et *copytask*. Avec ces deux éléments, HAMSTER offre la possibilité de faire référence à une autre tâche dans le modèle de tâches (*copytask*) ou de faire référence à un autre modèle de tâches (*subroutine*). Cependant, dans certains systèmes de contrôle-commande, les sous-tâches des tâches répétitives, peuvent changer (de nom) d'une situation (manipulation d'une commande) à une autre. Dans ce cas, les patterns de tâches sont mieux adaptés car ils permettent de décrire le nom des tâches en termes de variable à adapter à chaque contexte d'utilisation.

Les approches permettant de faciliter la conception des modèles de tâches complexes ayant été présentées, nous proposons maintenant de mettre en évidence les problématiques et besoins liés à leur application.

4. Problématique

La description en modèles de tâches des activités de l'utilisateur sur un système de contrôle-commande illustre bien la complexité de conception de ces modèles.

Dans la section 2 de ce chapitre, nous avons présenté quelques approches qui exploitent les patterns de tâches pour offrir une bonne solution de réutilisabilité et réduisent la complexité dans la conception des modèles de tâches. Ces approches proposent des outils basés sur l'utilisation des patterns de tâches stockés dans une base de données, pour la description des modèles de tâches. Les limites de ces outils se trouvent à deux niveaux. Tout d'abord, il est très difficile d'imaginer tous les patterns possibles pour les stocker dans un Framework. Les patterns conçus dans l'entrepôt ne sont pas adaptés à tous les domaines. Dans le livre de (Seffah 2015) au chapitre 5, les auteurs ont rencontré ce problème en voulant construire des modèles de tâches pour un *smart environment*. Ils ont dû construire un nouveau langage de patterns adapté à ce domaine. D'autre part, les outils existants ne sont applicables qu'à des modèles de tâches définis avec XIML au mieux, avec la notation CTT. Or l'utilisation des

notations comme CTT pour structurer l'activité complexe des opérateurs en supervision est ardue (Martinie 2011).

Certains des travaux présentés proposent des outils permettant de passer des patterns de tâches aux modèles de tâches. Ces outils sont généralement utilisés par les concepteurs d'IHM car ils renferment des notions encore complexes pour les utilisateurs. Une fois les modèles de tâches obtenus, ils devront être simulés avec les utilisateurs experts, qui possèdent les connaissances sur le fonctionnement du système, pour leur vérification et validation. Lors de la simulation des erreurs détectées peuvent conduire à d'importantes modifications dans les modèles obtenus. Ces modifications peuvent affecter le temps de conception du système de contrôle-commande.

La question que nous nous sommes posée est : comment pouvons-nous aider les utilisateurs experts (experts métiers) à obtenir les modèles de tâches, sans contrainte de notation ni de formalisme ?

Un des verrous identifié dans cette thèse concerne l'obtention des modèles de tâches complexes par utilisation des patterns de tâches. L'état de l'art ne permettant de traiter complètement ce verrou, il faut donc réfléchir, d'une part, à une démarche permettant de construire les patterns qui décrivent les tâches d'un opérateur face à un système de contrôle-commande. Cela nécessite une identification et une formalisation des dites tâches. D'autre part, il faut pouvoir exploiter au mieux les patterns définis ainsi que les connaissances des experts métiers pour réduire la complexité dans l'obtention des modèles de tâches.

5. Proposition d'une démarche de conception des modèles de tâches complexes

Nous tirons parti de l'analyse de l'activité présentée dans la section 1 de ce chapitre, pour construire un pattern de tâches afin de faciliter la conception des modèles de tâches complexes. Le passage par la conception des patterns pour aboutir aux modèles de tâches nous semble être une bonne solution. En effet, pour (Jacob, Limbourg, et Vanderdonckt 2005), les patterns de tâches décrivent de façon générique, les actions que l'utilisateur doit effectuer pour atteindre un certain objectif. La description de l'objectif agit comme une identification non ambigüe pour le pattern. Pour que le pattern soit le plus générique et le plus flexible possible, la description de l'objectif doit comporter au moins un composant variable. L'idée est de promouvoir la réutilisabilité et faciliter la définition des modèles de tâches des fonctions de haut-niveau.

Les travaux de (Breedvelt-schouten, M., Paternò, et Severijns 1997) sont axés sur les patterns de tâches. Les patterns de tâches encapsulent des *templates* de tâches pour les problèmes courants de conception. Plus précisément les patterns de tâches sont utilisés comme des *templates* (ou des blocs de conception des tâches) pour la construction du modèle de tâches d'une application. Selon Breedvelt, un autre avantage des patterns de tâches est qu'ils facilitent la lecture et l'interprétation de la spécification des tâches.

Bien que les patterns de tâches facilitent la définition des modèles de tâches, leur définition n'est pas souvent facile (Breedvelt-schouten, M., Paternò, et Severijns 1997). Il y a au moins deux aspects importants qui ressortent :

- Identification des situations caractéristiques qu'il peut y avoir dans différentes applications ;
- Identification des informations qui pourraient être utilisées pour définir les instances de ces patterns.

L'identification des informations contenues pour construire un pattern peut s'appuyer sur la formalisation des spécifications. De plus, la définition du pattern de tâches suit un formalisme bien défini pour en permettre la compréhension par les concepteurs d'IHM.

5.1. Formalisme de construction des patterns

Dans la littérature, il existe plusieurs formalismes de représentation des patterns. Certains sont spécifiques à la représentation des patterns de produits, d'autres à celle des patterns de processus. Cependant ces formalismes présentent certaines faiblesses : dispersion des informations, absences de formalisation des rubriques, limite des solutions et insuffisance des relations inter-patterns. Toutes ces informations sont bien détaillées dans (Conte et al. 2001). Pour combler ces lacunes, le formalisme P-Sigma a été proposé par (Conte et al. 2001). Ce formalisme est composé de trois parties : Interface, Réalisation et Relation. Il permet une représentation plus complète des patterns de conceptions et a été appliqué à plusieurs projets industriels.

Rubrique	Champ
Nom	Le nom du pattern
Problème	Définit le problème résolu par le pattern. *Exemple: il permet de construire un modèle de tâches de réalisation d'une fonction sur une IHM de contrôle commande de haut niveau.*
Contexte	Décrit la précondition pour l'application du patron. Peut être obtenu en appliquant la solution modèle d'un ou de plusieurs patrons ; les noms des patrons correspondants constituent alors le champ formel. *Exemple: Le pattern est applicable si le système de contrôle commande dispose de fonctions dont la réalisation nécessite une commande globale.*
Solution	Indique la solution du problème en terme de processus à suivre. *Exemple: Pour lancer une commande globale, l'utilisateur doit cliquer sur la fonction représentant la commande, il verra donc un bandeau s'afficher qui lui servira pour lancer la fonction. Une fois la fonction lancée il pourra à tout moment l'arrêter s'il le désire.*
Relation	Le ou les patterns au quel ce pattern est lié.
Représentation	Représentation graphique de la solution.

Figure 31 Formalisme de représentation des patterns de modèle de tâches inspiré de (Sinnig 2004)

Dans le contexte de conception des patterns de tâches, le formalisme TPML a été décrit lors du workshop CHI 2003. Le travail présenté dans ce mémoire concerne les patterns de tâches. Nous choisissons donc d'utiliser ce formalisme.

Les rubriques classiques telles que Nom, Problème, Contexte, Solution et Relation sont décrits pour permettre au concepteur de comprendre à quoi sert le pattern. La représentation graphique, quant à elle, permet de représenter la structure arborescente d'un pattern.

5.2. Représentation graphique du pattern de tâches

La structure de la tâche (de haut niveau) de pilotage d'un système de contrôle commande peut être résumée suivant certaines phases. On peut décrire un pattern de tâches contenant toutes les phases ainsi que leur décomposition hiérarchique. Cependant, essayer de décrire une tâche complexe en un seul pattern peut être très difficile, et la description peut vite devenir très lourde (Granlund, Lafrenière, et Carr 2001). L'intérêt premier de l'utilisation des patterns de tâches est de rendre la spécification de la tâche plus facile à lire et à interpréter (Breedvelt-

schouten, M., Paternò, et Severijns 1997), car il est possible d'indiquer leurs noms si elles ne sont pas détaillées dans l'arbre de tâches plutôt que de les détailler complètement. Cependant, il devient incontournable de définir les tâches complexes en un seul pattern si les sous-patterns identifiés ne peuvent être réutilisés dans des contextes autres que la supervision d'un système de contrôle-commande.

La représentation graphique simple du pattern de tâches de supervision tels que présenté dans (Bovell, Carter, et Beck 1998) permet juste de voir l'arborescence des tâches, mais on il ne présente aucune information sur la décomposition des tâches, sur les opérateurs, sur les exécutants, etc. Le langage TPML et ses variantes permettent de représenter les patterns graphiquement et textuellement mais l'instanciation des patterns se fait avec des outils spécifiques qui exploitent ces langages. Un modèle de tâches qui intègre ces patterns doit être décrit soit en CTT, soit en XIML. Les autres langages de modélisation des tâches ne sont pour le moment pas pris en compte. Il est préférable de pouvoir décrire les patterns à partir de n'importe quel langage de modélisation de tâches.

Nous avons choisi d'utiliser pour nos travaux l'outil de modélisation de tâches K-MADe pour décrire le pattern de tâches de supervision. L'intérêt de cette représentation est de pouvoir prendre en compte, déjà au niveau du pattern, les informations concernant la décomposition des tâches, les opérateurs, les exécutants, les préconditions etc. L'idéal est d'utiliser directement les patterns sans passer par des procédures de transformations d'un langage de patterns à un langage de représentation de modèles de tâches. Cependant les procédures de transformations sont inévitables si on veut passer d'une notation de tâches à une autre.

Les informations relatives à un pattern telles que Problème, Contexte, Solution et Relation peuvent être décrites dans la section « Description » du nom de la tâche principale du pattern. Ces informations aident le concepteur à comprendre le pattern de tâches.

Nous décrivons ci-dessous la conception du pattern de tâches en l'illustrant avec un système de contrôle-commande du domaine naval. Ce pattern peut, bien sûr, être adapté à d'autres domaines.

La supervision d'un système de contrôle-commande (« *Superviser le système* », Figure 32) s'effectue en quatre grandes tâches qui sont réalisées en parallèle par les opérateurs de la salle ou passerelle de contrôle : « *Surveiller les fonctions du système* », « *Détection et traitement de défauts* », « *Manipulation de commandes prévues* », « *Tâches administratives* ».

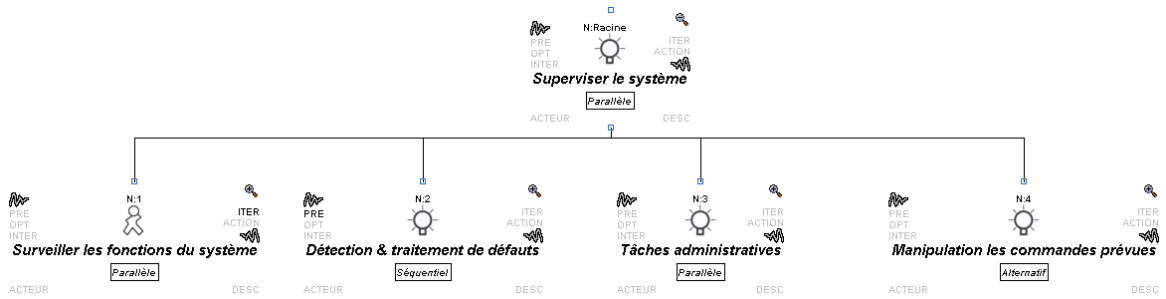


Figure 32 Description de la tâche « Superviser le système avec K-MADe

La tâche « *Surveiller les fonctions du système* » (Figure 33) est une tâche utilisateur itérative car elle s'effectue en permanence. Au cours de cette tâche l'opérateur doit surveiller en parallèle les paramètres du système, les variables du procédé, tendances et conditions anormales pour

signaler à l'équipage dès qu'il y a une anomalie détectée afin que la procédure appropriée soit lancée.

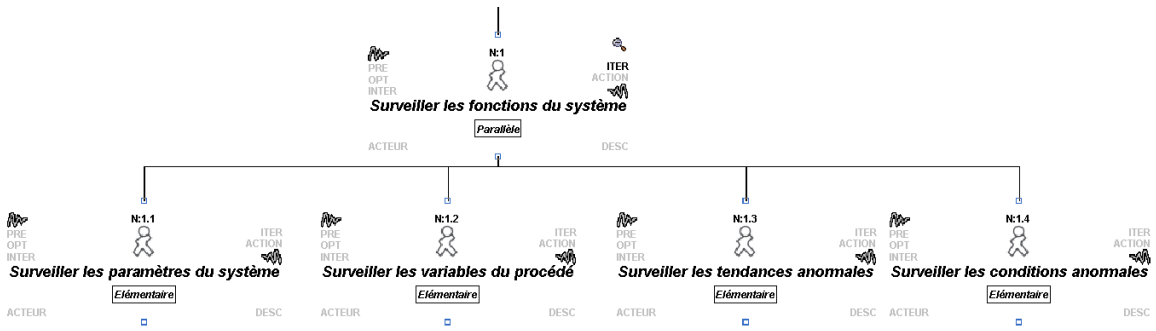


Figure 33 Représentation graphique de la sous-tâche « Surveiller les fonctions du système avec K-MADe

La tâche « Détection et traitement de défauts » (Figure 34) est une tâche système qui peut intervenir à n'importe quel moment de la supervision du système. En effet, le système peut remonter à tout moment une ou plusieurs alarmes liées aux composants et d'autres anomalies liées aux conditions, aux variables, etc. Les actions à effectuer par les opérateurs diffèrent en fonction des défauts détectés. Par exemple en cas d'accident (Définition 1), les opérateurs doivent effectuer des actions correctives immédiates pour régulariser la situation ou mettre le système en configuration de « Repli » (Définition 2). S'il n'y a pas d'accident, l'opérateur doit tout d'abord analyser le problème, puis sélectionner des alternatives en s'appuyant sur son expertise et les procédures prédéfinies pour restaurer le système. La sélection d'une alternative suit une séquence d'action bien définie. L'opérateur doit tout d'abord identifier les actions possibles permettant de corriger le problème, puis analyser les conséquences de ces actions.

La configuration de replis est la configuration à atteindre en cas de défaillance grave du système (grande fuite d'eau par exemple, sur un système de gestion d'eau). Pour les systèmes de gestion de fluide par exemple, elle consiste à refermer toutes les vannes et à arrêter tous les équipements (pompes, osmoseurs…).

Définition 2 Configuration de Replis

Ensuite il doit comparer les avantages et les inconvénients de ces actions pour enfin choisir l'alternative la mieux adaptée. L'état du système affiché après le traitement des défauts permet à l'opérateur d'avoir la confirmation que ces actions ont bien été prises en compte par le système et ont permis de restaurer ce dernier.

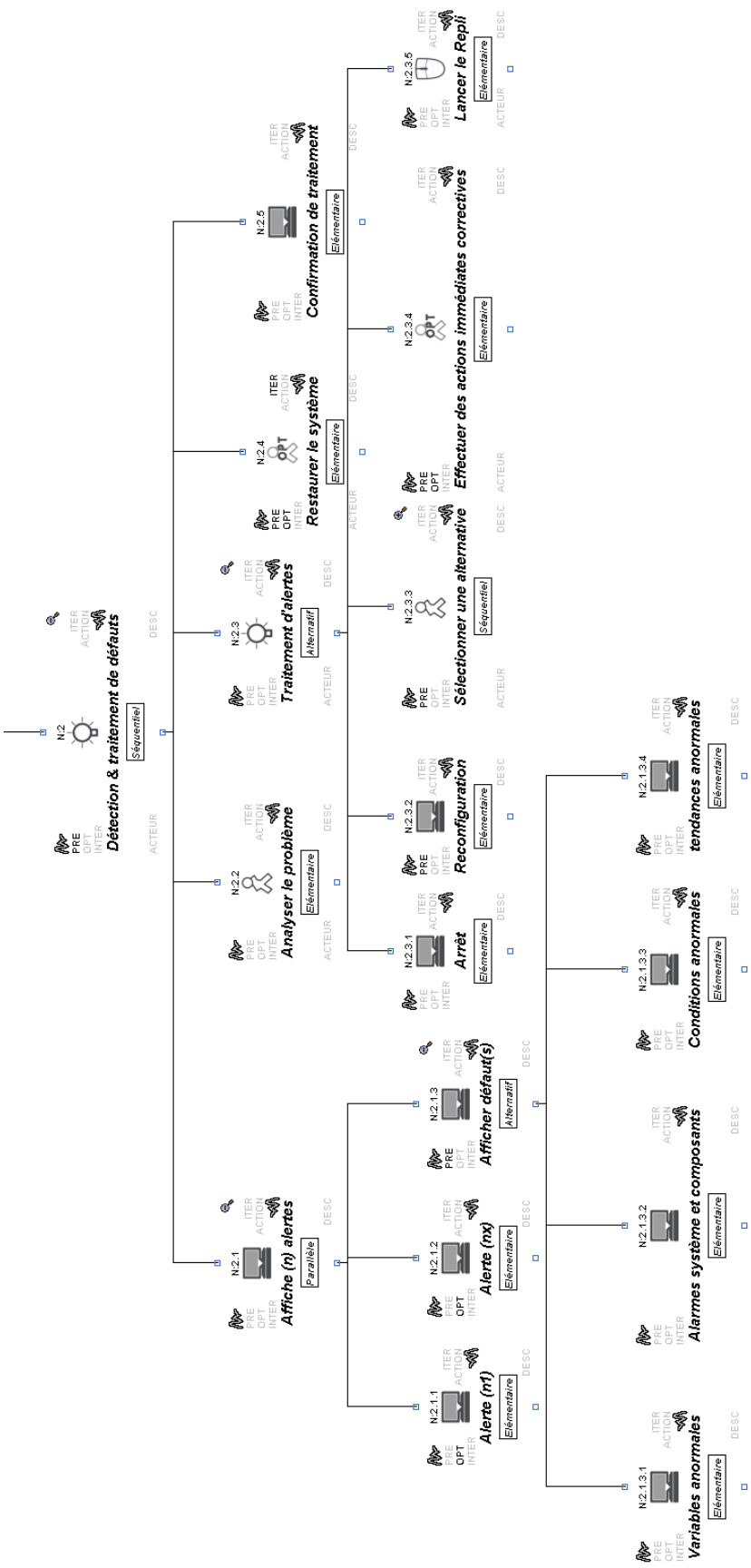


Figure 34 La sous-tâches « Détection et traitement de défauts » avec K-MADe

Les « *Tâches administratives* » (Figure 35) s'effectuent généralement par les opérateurs qui ne sont pas chargés de surveiller le système. Certaines de ces tâches telles que les tests de surveillance et la vérification des spécifications techniques ont pour but d'assurer la sécurité des installations en cas d'accident. D'autres telles que la tenue du journal et l'écriture des rapports d'évènement ont pour but de noter les problèmes majeurs détecté sur le système, les évènements avec risque d'aggravation, les évènements qui nécessitent une classification d'urgence, etc. Les rapports précis d'évènement permettent d'identifier les défauts de conception des installations, les procédures d'exploitation, l'instrumentation et la planification d'urgence (Bovell, Carter, et Beck 1998).

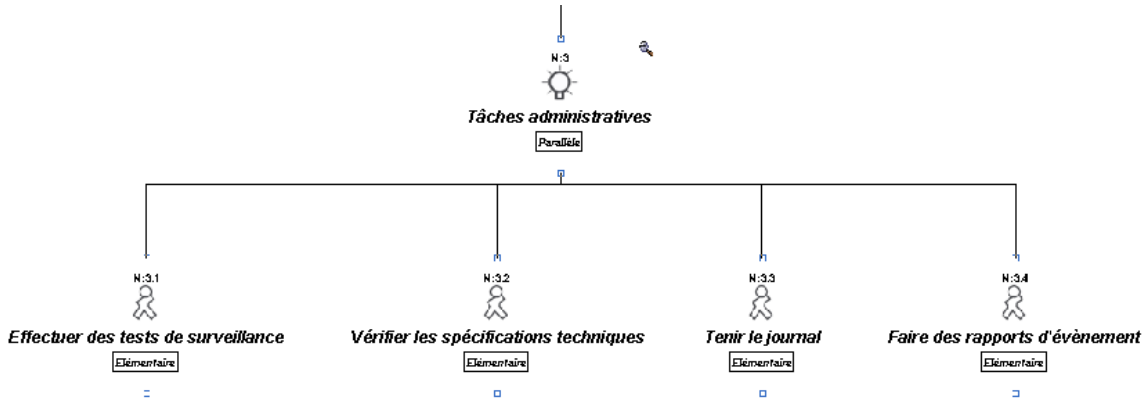


Figure 35 La sous-tâche « tâches administratives »

La tâche « *Manipulation de commandes prévues* » (Figure 36) est, quant à elle, une tâche abstraite permettant à l'opérateur de réaliser de façon semi-automatique les fonctions da haut-niveau du système et/ou de lancer toute autre commande prévue. Sur un grand nombre de systèmes de contrôle-commande, la plupart des manipulations de commandes se produisent soit au démarrage du système, soit lorsqu'il y a des arrêts normaux ou des évènements anormaux. Par exemple, à la fin de l'exécution d'une fonction qui permet de transférer de l'eau d'une soute à une autre dans un système d'eau douce, l'opérateur peut lancer une autre fonction selon l'état du système et les besoins des utilisateurs.

Les commandes prévues sont effectuées dans une séquence d'actions bien définie suivant une procédure écrite, avec des réactions du système à ces actions. Pour les commandes qui permettent de lancer une fonction, l'opérateur peut soit « *Effectuer une fonction* » si elle n'est pas en cours, soit « *arrêter une fonction* » qui est en cours. Pour effectuer une fonction, il doit cliquer sur la commande de la fonction pour la démarrer, ensuite il doit renseigner la consigne liée à la fonction en suivant une séquence d'actions bien définie. Une fois que l'opérateur a validé la consigne, l'automate local prend en compte la disponibilité des éléments (défauts, position, etc.), la disponibilité du système (les éventuels défauts des éléments et instrumentation) afin de déterminer si la demande effectuée par l'opérateur sera *acceptée*. Si cette demande est refusée, un message informe l'opérateur. Sinon le système se configure et l'opérateur peut évaluer en réalisant en parallèle les tâches « *Surveiller les fonctions du système* » et « *Détection et traitement de défauts* » décrites ci-dessus.

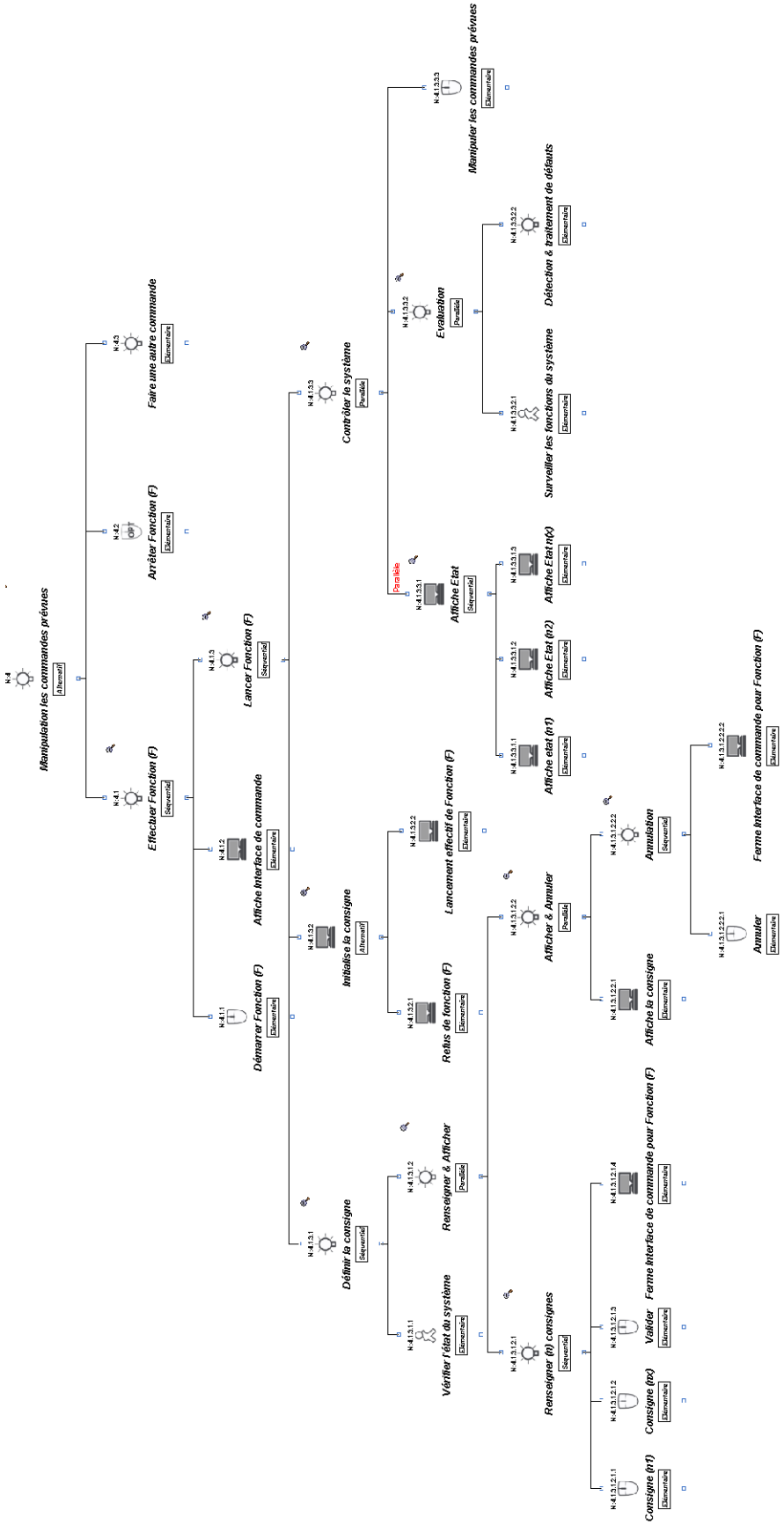


Figure 36 La Sous-tâche manipulation de commandes prévues avec K-MADe

L'annexe 2 présente l'ensemble du pattern de tâches « *Superviser le système* » issu de notre formalisation. La généricité de ce pattern de tâches est visible par la présence de variable dans la description de l'activité de supervision. Par exemple, dans l'annexe 2, on retrouve les noms de tâches tels que *consigne (nx)*, *affiche état (nx)* et *alerte (nx)*, qui doivent être adaptés aux différents contextes. La description des activités de supervision dans le pattern de tâches permet de recueillir, de façon globale et plus précise, des informations sur l'utilisation de système nécessaire à la mise en œuvre des spécifications fonctionnelles à travers les commandes de haut-niveau.

5.3. Du Pattern de tâches aux modèles de tâches

Le pattern de tâches ainsi décrit doit être adapté selon la fonction et le système dont on veut construire le modèle de tâches. Les informations permettant d'adapter correctement le pattern de tâches après l'avoir instancié sont connues par un utilisateur expert (expert métier) qui n'a aucune connaissance des notations de tâches. Il est donc préférable d'offrir la possibilité à cet expert d'adapter le pattern. Pour ce faire, le pattern de tâches doit être présenté sous une forme compréhensible, lui permettant de le lire, le parcourir et de l'adapter facilement. Notre objectif est de proposer une démarche proche de celle du simulateur de tâches Prototask (Lachaume et al. 2012) qui simplifie la vérification et la validation des modèles de tâches aux utilisateurs. Prototask est lié au formalisme de modélisation de tâches K-MAD. Pour exploiter pleinement les fonctionnalités de Prototask, la démarche de description du pattern de tâches que nous proposons s'effectue également avec K-MADe. Cependant, il est intéressant de rendre la démarche générique en n'obligeant pas le concepteur à utiliser une notation de pattern spécifique. Pour cela, une étape importante avant l'obtention des modèles de tâches est de transformer le pattern de tâches écrit dans d'autres notations de modélisation de tâches en K-MAD.

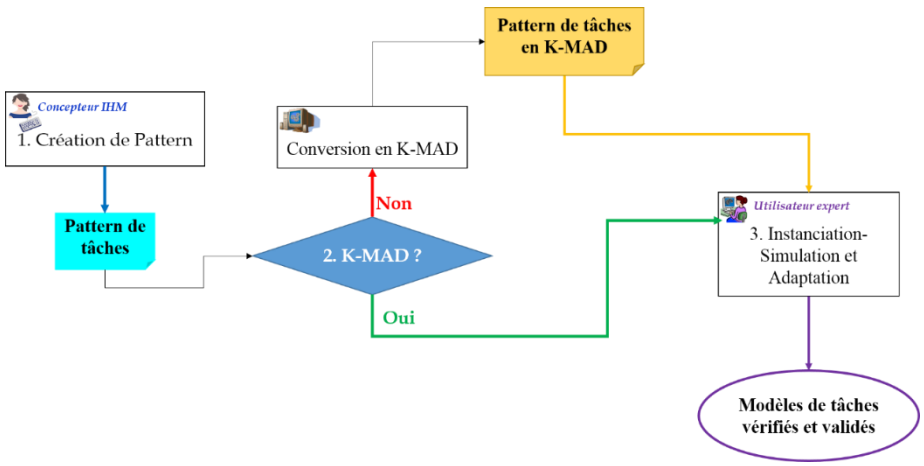


Figure 37 Démarche globale d'obtention des patterns à partir des modèles de tâches

La Figure 37 décrit la démarche globale de conception du pattern de tâches et d'obtention des modèles de tâches. La première étape de cette démarche est la conception du pattern de tâches (Annexe 2) de supervision par le concepteur d'IHM. Ce pattern suit le formalisme et la représentation graphique présentés respectivement dans les sections 5.1 et 5.2. Il peut être réutilisé pour la conception des modèles de tâches de la plupart des systèmes de contrôle-commande. Cette étape est suivie de la vérification de la notation dans laquelle est décrit le pattern. Si le pattern est décrit en K-MAD, il est ensuite instancié au cours de la deuxième étape, dans un outil d'adaptation de pattern de tâches pouvant permettre à un utilisateur

expert (expert métier) de l'adapter aux différents contextes d'utilisation (aux spécificités de la fonction dont on veut décrire le modèle de tâches). Si le pattern n'est pas décrit avec K-MADe, une étape intermédiaire permet de le convertir en un modèle K-MAD afin de l'utiliser dans le processus d'adaptation. L'outil proposé, dont l'implémentation est détaillée dans le chapitre 4, offre la possibilité à l'utilisateur expert d'adapter le pattern de tâches tout en le simulant pour obtenir les modèles de tâches vérifiés et validés.

6. Bilan sur la conception des modèles de tâches complexes

Les travaux présentés dans ce chapitre, apportent une aide supplémentaire par rapport aux approches existantes, pour la conception des modèles de tâches complexes.

La formalisation, la construction et l'utilisation de patterns de tâches nous permettent de résoudre les problèmes de complexité et de diversité qui augmentent dans la description des modèles de tâches. En effet, l'identification de la structure de la tâche de supervision en s'appuyant sur l'état de l'art a conduit à la construction d'un pattern regroupant, de façon générique, les activités des opérateurs d'une salle ou une passerelle de contrôle.

L'utilisation d'un logiciel concret de modélisation de tâches (K-MADe) pour la représentation du pattern permet une meilleure description de ces derniers. De plus, nous offrons ainsi la possibilité au concepteur de construire lui-même le pattern dont il a besoin pour l'analyse des activités humaines. Pour la généricité de notre approche, nous tentons de nous appuyer sur le principe PIM (Plateform Independant Model), c'est-à-dire les patterns dont la description contient des variables, peuvent être spécifiés avec des outils de modélisation autre que K-MAD. Les concepteurs d'IHM peuvent donc représenter ces patterns avec n'importe quelle notation de modélisation de tâches (KMAD, CTT, Hamsters, etc…). Il faudra dans ce cas assurer le passage d'un modèle à l'autre.

La mise en œuvre des transformations de modèles permet de s'affranchir des contraintes liées au choix de la notation de tâches. Cette opération peut nécessiter d'importantes transformations de modèles mais ces transformations ne sont mises en œuvre qu'à la première utilisation d'une nouvelle notation de tâches.

La proposition d'une démarche d'adaptation qui s'appuie sur la méthode Prototask permet de rendre l'adaptation et la lecture du pattern de tâches accessible aux utilisateurs experts et d'avoir des modèles de tâches vérifiés et validés.

Les modèles de tâches ainsi obtenus contiennent les exigences formalisés nécessaire à la conception du système de contrôle-commande. Ces exigences devront être complétées par les spécifications fonctionnelles du système. En effet, les modèles de tâches permettent d'avoir un ensemble d'exigences fonctionnelles exprimées par le concepteur qui peuvent être utilisées dans les approches de conception, mais ces modèles ne permettent pas d'éviter les erreurs liées à l'interprétation des spécifications fonctionnelles.

Chapitre 3 : Spécifications et Génération d'application de contrôle-commande

L'analyse et la modélisation de la tâche humaine sont largement intégrées que ce soit dans le processus de conception ou celui de génération des systèmes interactifs, car elles permettent d'extraire un ensemble de données relatives aux besoins de l'utilisateur (besoins informationnels de l'opérateur (BIO)) qui sont transformées et intégrées aux IHM. Cependant, le formalisme de modélisation de la tâche, la validation du modèle de tâches et le passage du modèle de tâches vers ces besoins et les objets de l'interface sont autant d'éléments qui rendent difficile la prise en compte de l'analyse de la tâche dans une approche de génération de système interactif (MOUSSA 2005).

Pour résoudre ce problème, l'analyse de la tâche est souvent combinée avec les spécifications fonctionnelles du système. Cependant l'état de l'art présenté au chapitre 1 rend bien compte des problèmes liés à la spécification fonctionnelle des systèmes complexes (voir section 3.6 au Chapitre 1).

Dans ce chapitre, nous tentons d'apporter une solution aux difficultés rencontrées dans la spécification fonctionnelle des systèmes complexes pour la génération de systèmes de contrôle-commande.

La *première section* de ce chapitre présente un état de l'art sur les approches permettant de générer les systèmes interactifs à partir des modèles de tâches. Cette première section nous permet d'identifier des verrous présentés dans la deuxième section de ce chapitre. Ces verrous mettent en évidence le besoin de compléter le modèle de tâches par d'autres modèles : les spécifications fonctionnelles. Il apparaît alors pertinent de chercher des pistes pouvant faciliter l'obtention de ces spécifications.

Ainsi, la *deuxième section* de ce chapitre présente un état de l'art des approches permettant de capturer la connaissance que les concepteurs ont du système à concevoir pour obtenir ainsi les spécifications fonctionnelles.

En tirant parti de cet état de l'art, nous proposons une démarche détaillée dans la *troisième section* de ce chapitre, dont le but est de faciliter la spécification fonctionnelle des systèmes complexes.

1 La génération d'IHM à partir de modèles de tâches

Depuis quelques années, un grand nombre d'outils et de méthodologies exploitent les modèles de tâches pour la génération d'interface. Une interface utilisateur (UI) est la partie d'un système logiciel/matériel qui est conçue pour permettre l'interaction avec des utilisateurs (Ramón et al. 2015). Dans les approches WIMP (Windows, Icons, Menus and Pointers), qui constituent encore la majorité d'interfaces sur ordinateur[15], elle est présentée avec des éléments graphiques tels que des icônes ou des menus. Les interfaces utilisateurs disposent d'une partie statique qui est liée à la présentation (par exemple la structure, la disposition, l'accès ou l'esthétique) et d'une partie dynamique qui est liée au comportement du système par rapport aux actions de l'utilisateur (c'est-à-dire les événements qui sont déclenchés et l'exécution des

[15] Nous ne considérerons pour notre travail que les interfaces WIMP, seules utilisées aujourd'hui dans les systèmes industriels qui sont cibles de nos travaux.

actions et/ou des changements). Pour la suite du chapitre nous nommons la partie statique, *présentation* et la partie dynamique, *dialogue*. Nous définissons ces deux termes avant de présenter un état de l'art sur la génération d'interface à partir de modèles de tâches.

1.1 La présentation

Le modèle de présentation d'une interface peut être considéré comme une sorte de disposition spatiale des éléments graphiques (widgets ou vues) dans la vue de l'application (Ramón et al. 2015). Les vues sont les représentations graphiques qui sont affichées sur les écrans du dispositif sur lesquels l'interface tourne (les fenêtres dans les applications de bureau, des pages web dans les applications web, des vues dans les applications mobiles etc.).

1.2 Le dialogue

Selon (Luyten et al. 2003), on identifie trois différents niveaux de dialogue : le *dialogue inter-fenêtre*, le *dialogue intra-fenêtre,* et le *dialogue intra-widget*. Un modèle de dialogue est un modèle abstrait utilisé pour décrire la structure du dialogue entre un utilisateur et un système interactif (Green 1986). Plusieurs notations sont utilisées pour décrire le modèle de dialogue où le système est représenté comme un ensemble d'états liés par des transitions. Le modèle de tâches peut, au choix des concepteurs, représenter concrètement le dialogue à chacun des trois niveaux, mais aussi au niveau complètement abstrait.

1.2.1 Le dialogue inter-fenêtre

Le *dialogue inter-fenêtre* définit le comportement des différentes présentations (fenêtres) les unes avec les autres. Par exemple sur la Figure 38, le clic sur le bouton « Afficher » de la fenêtre N°1, permet d'ouvrir la fenêtre N°2.

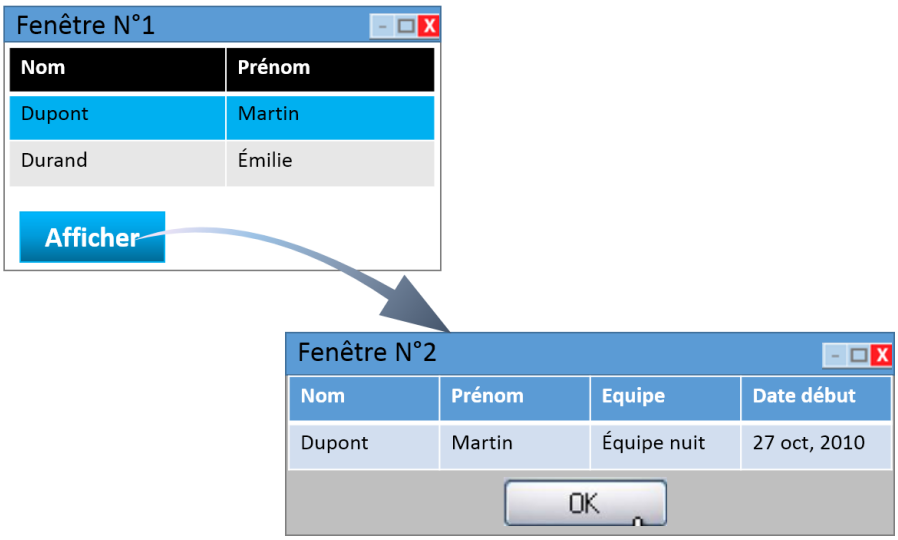
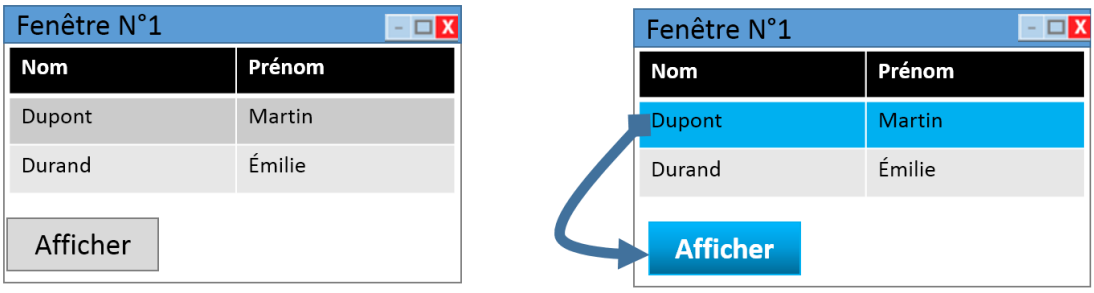


Figure 38 Exemple de dialogue inter-fenêtre

La complexité des dialogues homme-machine a conduit de nombreux travaux à s'intéresser à leur formalisation et à leur vérification (Caffiau 2010). Les outils de simulation développés pour les modèles de tâches démontrent le lien naturel entre la dynamique exprimée dans les modèles de tâches et le modèle de dialogue implémenté dans l'application correspondante.

1.2.2 Le dialogue intra-fenêtre

Le *dialogue intra-fenêtre* définit le dialogue des widgets dans une même présentation (fenêtre de l'application). Ce dialogue correspond à la réponse du système par rapport aux autres widgets de la fenêtre lorsque l'utilisateur réalise une tâche.



Avant la sélection de la ligne – Bouton « *Afficher* » inaccessible

La sélection de la ligne : Bouton « *Afficher* » accessible

Figure 39 Exemple de dialogue intra-fenêtre

La Figure 39 illustre un exemple de dialogue intra-fenêtre. Sur la Figure 39a, illustrant la fenêtre de l'application à l'ouverture, le bouton « Afficher » est inaccessible (l'utilisateur ne peut pas cliquer dessus). La Figure 39b illustre la réaction de l'application lorsque l'utilisateur sélectionne une ligne du tableau, il peut alors cliquer sur le bouton « Afficher » qui est devenu accessible après sa sélection.

1.2.3 Le dialogue intra-widget

Le *dialogue intra-widget* définit comportement du widget, qui n'est généralement pas modifié dans les interfaces WIMP (Windows, Icons, Menus and Pointing device). En effet, Le principe du style d'interaction WIMP, c'est d'utiliser les widgets sans les modifier, pour standardiser les comportements. Par exemple, si l'utilisateur clique sur un widget type de « liste déroulante », le système doit afficher la liste des informations proposées par ce widget pour lui permettre de faire un choix. Ce dialogue décrit donc les actions de l'utilisateur lorsqu'il manipule un widget pour réaliser une tâche et la réponse du système lors de cette manipulation.

1.3 Les approches de génération d'interface à partir de modèle de tâches existantes

Dans la littérature, certains travaux ont proposé de générer toute ou partie d'une interface à partir des modèles de tâches. D'autres ont combiné le modèle de tâches avec d'autres modèles tels que le modèle de l'utilisateur, le modèle de domaine, etc., pour générer l'interface utilisateur. Nous proposons dans cette section, un panorama non exhaustif des approches existantes.

1.3.1 Les travaux de Paterno et al, 1999 – Deriving presentation from task models

La motivation derrière les travaux de (Paterno, Breedvelt-schouten M., et Koning 1999) est de faciliter l'utilisation des modèles de tâches dans la conception des systèmes interactifs. Les auteurs partent du constat que la plupart des approches basées sur les tâches s'intéresse à l'analyse et la conception du dialogue des applications interactives. Peu de travaux (Vanderdonckt et Gillo 1994; Zhou et Feiner 1997) s'étaient jusqu'alors intéressés à la façon dont un modèle de tâches peut être utilisé pour construire la présentation de l'application. La

conception de la présentation commençait trop tard, c'est-à-dire une fois le modèle d'architecture défini.

Le modèle de tâches comporte implicitement les informations dont a généralement besoin pour construire les systèmes interactifs. Il peut donc être utilisé pour analyser et déduire des indications pour la conception de la présentation d'une interface utilisateur. L'approche proposée ici est intégrée à CTTE (Concurrent Task Tree Environment) et consiste à faire une analyse descendante du modèle de tâches exprimé en utilisant la notation CTT (Concurrent Task Tree), pour indiquer comment concevoir la présentation. Le choix de l'analyse descendante est justifié par le fait qu'une analyse ascendante de l'arbre de tâches peut générer une conception non-cohérente qui nécessiterait de nombreuses modifications ultérieures. Ainsi, dans les approches ascendantes, les concepteurs associent une présentation possible à chaque tâche basique, puis ils composent ces présentations basiques pour obtenir la présentation générale de l'interface utilisateur. En étudiant l'approche ascendante, les auteurs s'étaient rendus compte que l'association d'une présentation à une tâche basique par rapport à la conception des autres tâches basiques peut conduire à des présentations peu cohérentes et conflictuelles ; à l'inverse, avec les approches descendantes, il est possible dans un premier temps de prendre les décisions de conception d'ensemble, puis d'affiner chaque présentation basique dans un système commun. Les auteurs décrivent le passage du modèle de tâches à l'obtention de la présentation comme suit :

- Identification des ensembles d'activation (ensemble de tâches activables dans un même laps de temps) à partir du modèle de tâches.
- Identification de la structure principale de présentation de chaque ensemble d'activation.
- Identification des transitions entre les présentations de deux ensembles d'activation.
- Identification du Template de présentation.
- Identification des patterns de présentation.
- Génération de la présentation de l'interface utilisateur.

En suivant ces différentes phases, les auteurs proposent un outil intégré à l'environnement CTTE, qui permet de générer automatiquement les présentations d'interface à partir de modèle de tâches décrit en CTT, des Templates de tâches basiques, des patterns de tâches prédéfinies et des règles décrites dans l'outil.

1.3.2 Les travaux de Berti et al. 2003 – An Environment for Designing and Developing Multi- Platform Interactive Applications.

Les travaux réalisés par (Berti et al. 2003) consistent à utiliser les modèles de tâches et les modèles des utilisateurs (Brusilovsky 1998) pour générer les interfaces des applications multiplateformes. La modélisation des utilisateurs permet d'adapter les interfaces à plusieurs types de dispositifs (Figure 40) et ce sur différentes plateformes pendant la phase de conception. Une plate-forme, au sens large, est définit comme un environnement qui permet de gérer et/ou d'utiliser les services d'une application. Les modèles de l'utilisateur décrivent des informations telles que les connaissances de l'utilisateur, ses préférences personnelles, son intérêt, ses déplacements et l'environnement.

L'idée de base de ces travaux est de générer des interfaces à partir des modèles de tâches (Figure 40). Ils extraient donc toutes les exigences pertinentes au niveau du modèle de tâches et ensuite ils utilisent ces informations pour générer des interfaces utilisateur efficaces,

adaptées aux différents types de plates-formes renseignées. Les informations sur la conception de l'interface peuvent être déduites de l'analyse des modèles de tâches. Par exemple, la décomposition logique d'une tâche peut guider la génération de l'interface correspondante. La structure d'une tâche est reflétée dans la représentation graphique par le regroupement des techniques d'interactions et des objets associés à une même sous-tâche. Les auteurs ont ainsi identifié un certain nombre de possibilités pour l'utilisation des modèles de tâches pour générer des interfaces utilisateurs multiplateformes ; par exemple :

- quand la même tâche peut être réalisée de la même façon sur plusieurs plateformes ;
- quand la même tâche est réalisée sur plusieurs plateformes mais avec différents objets de l'IHM.
- quand les tâches ne sont significatives que sur des plateformes spécifiques.

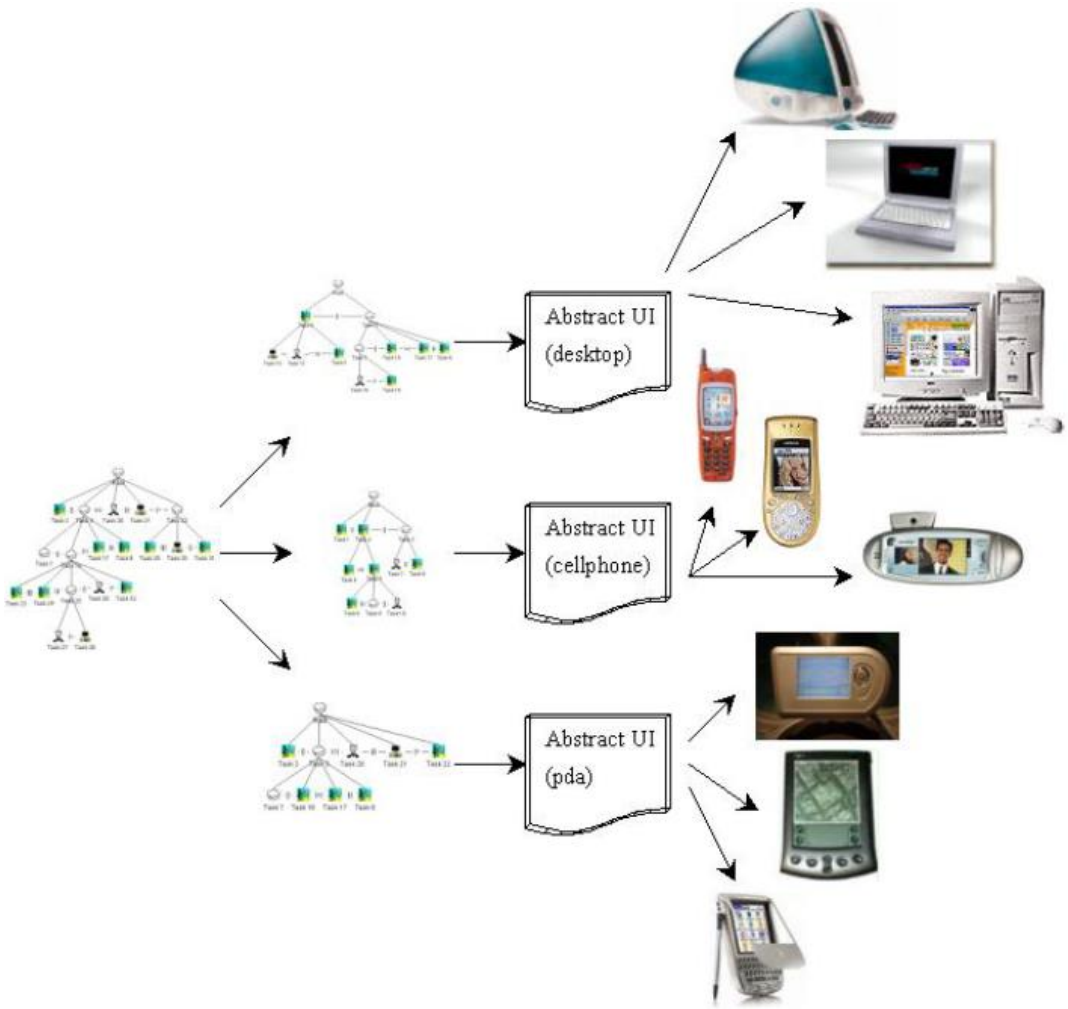


Figure 40 Les transformations supportées par TERESA (Berti et al. 2003)

Ces travaux ont conduit à la proposition d'un outil, TERESA (Transformation Environment for inteRactivE Systems representAtions). TERESA est un environnement semi-automatique développé pour générer une interface utilisateur concrète, pour un type spécifique de plateforme, à partir des modèles de tâches. Le passage du modèle de tâches à l'interface concrète se fait dans TERESA à travers quatre principales étapes. La *première étape* consiste à créer un modèle de tâches de haut niveau décrivant une application multi-contexte. La *deuxième étape* consiste à développer un modèle de tâches pour chaque plateforme considérée. La *troisième étape* utilise le modèle de tâches de chaque plateforme pour générer une interface

abstraite composée d'un ensemble d'interacteurs abstraits. La *dernière étape* consiste à créer une plateforme graphique indépendante (interface concrète) en faisant correspondre les interacteurs abstraits à des techniques d'interaction de la plateforme spécifique. La Figure 40 illustre les résultats obtenus à chaque étape.

Le modèle de tâches, qui est le modèle d'entrée de TERESA, est décrit avec la notation ConcurTask Trees (CTT). La notion d'ETS (Enabled Task Set), qui correspond à l'ensemble des tâches activables à tout moment du déroulement de la tâche, est essentielle à la génération de l'automate qui représente la dynamique de l'interface. En plus des informations classiques que l'on trouve dans tous les modèles de tâches, TERESA permet de spécifier les objets qui doivent être manipulés pour l'accomplissement de chaque tâche (il est possible de considérer à la fois les objets de l'interface utilisateur et ceux du domaine), ainsi qu'un certain nombre d'attributs supplémentaires, telle que la fréquence, qui seront utilisées pour la génération. Cette dernière s'effectue de façon semi-automatique, TERESA proposant au concepteur de choisir entre plusieurs solutions (type de widget ou disposition des widgets, par exemple) au fur et à mesure de la génération des différentes fenêtres.

1.3.3 Les travaux de Luyten 2004 – Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development.

Les travaux de (Luyten 2004) font suite à ceux décrits dans (Luyten et al. 2003) et ont conduit à la réalisation d'un framework, Dygimes (DYnamically Generating user Interfaces for Mobile computing devices and Embedded Systems). Le but de ce framework est de permettre la génération automatique des interfaces pour une même application pouvant tourner sur différentes plateformes. La différence avec TERESA se trouve dans le processus de génération. TERESA implique l'utilisateur par son modèle ce qui fait que la génération des interfaces est semi-automatique. Du fait que la génération dans Dygimes framework est automatique, cet outil offre un environnement de développement moins évolué que TERESA qui propose une génération semi-automatique.

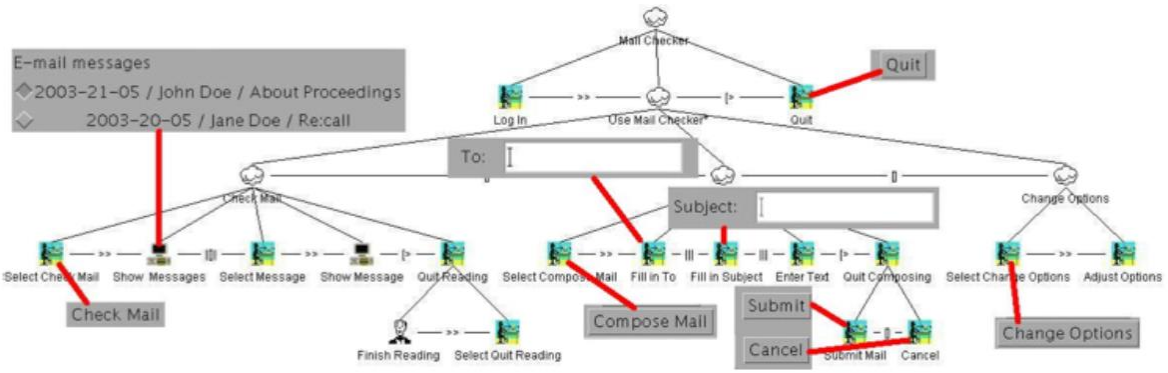


Figure 41 Exemple d'un diagramme CTT annoté avec les widgets associés aux tâches (Caffiau 2009)

Tout comme TERESA, Dygimes se base sur un diagramme CTT (Figure 41) pour générer différentes présentations (Figure 42) et le dialogue (Figure 43) d'une application. Lors de la conception du modèle de tâches avec la notation CTT, les concepteurs associent des widgets à certaines tâches élémentaires, par exemple, un bouton « Cancel » qui est associé à une tâche « Cancel ».

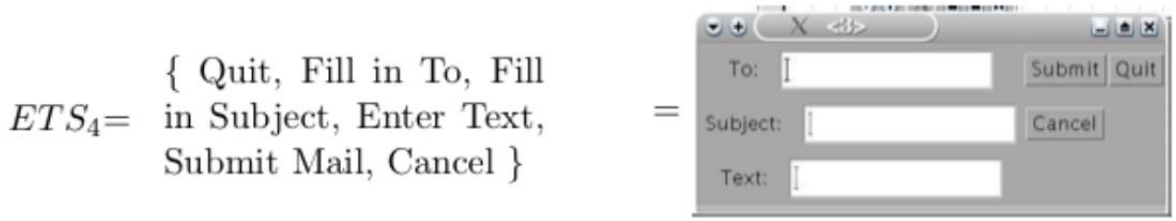


Figure 42. Exemple de la représentation d'état graphique (Caffiau 2009).

Comme dans TERESA, cette approche permet de réaliser le modèle de dialogue (Figure 43) en calculant les ETS afin de former les états, puis les transitions sont déterminées parmi les tâches de chaque ETS. Cependant, dans Dygimes, la transformation du modèle de tâches décrit dans le formalisme CTT en description du dialogue sous forme d'automate est détaillée alors que l'outil TERESA ne met à notre disposition que des fichiers XML générés.

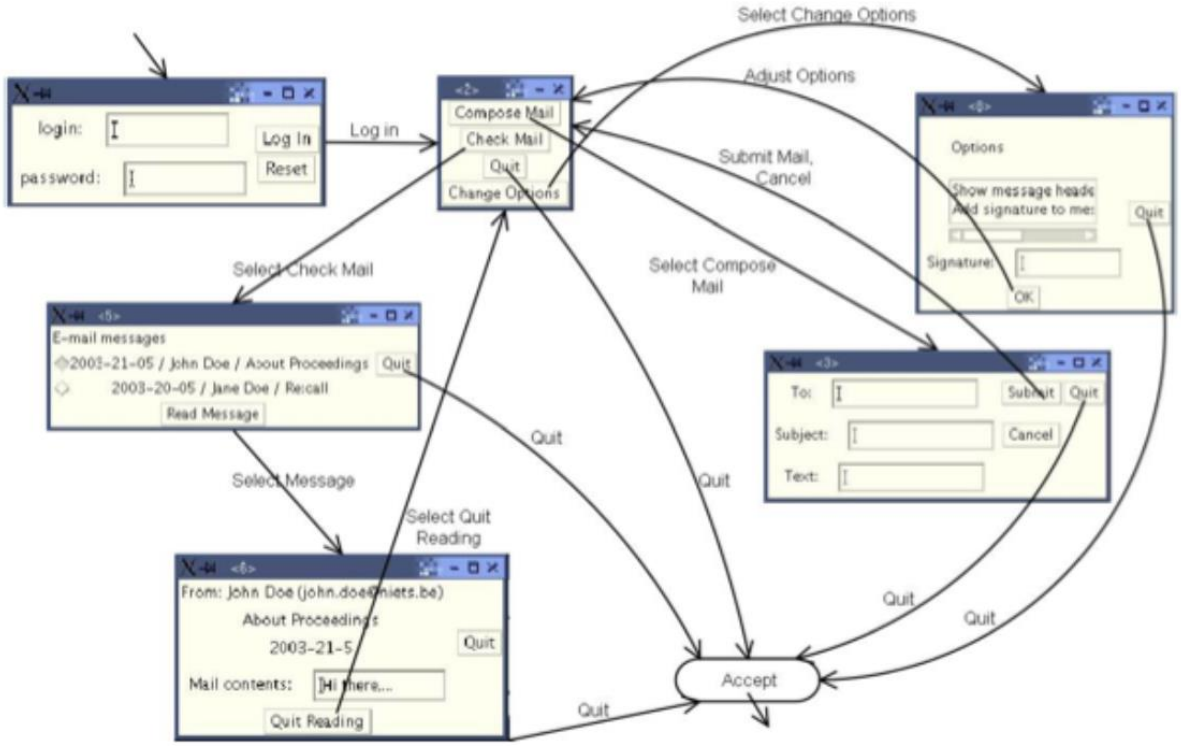


Figure 43 Le dialogue de l'application exemple (Caffiau 2010)].

Une des principales limitations de Dygimes réside dans le choix des auteurs de ne prendre en compte que le dialogue inter-fenêtre, sans se préoccuper de ce qui peut se produire dans une fenêtre (activation/désactivation d'un bouton, par exemple).

1.3.4 Les travaux de Wolf et Forbrig, 2009 – Deriving user interface from task models

Les travaux de (Andreas Wolff et Forbrig 2009) permettent de dériver une interface utilisateur à partir des modèles de tâches. Les auteurs sont partis du constat que la génération automatique de graphes de dialogue à partir de modèles de tâches n'est pas une tâche triviale. TERESA utilise ainsi des heuristiques pour déduire un modèle de dialogue du modèle de tâches, mais elles n'ont pas été publiées. Certaines approches basiques sont évidentes, comme par exemple « mettre chaque tâche dans une vue » ou bien « mettre toutes les tâches dans une vue unique ». Ces approches ne sont cependant pas satisfaisantes.

Pour la génération de l'interface utilisateur, les auteurs ont proposé une démarche semi-automatique qui n'intègre pas le calcul des ETS. La méthodologie consiste à fusionner les modèles de tâches avec un modèle de navigation pour dériver les modèles d'interface abstraite et concrète, et finalement générer du code source. Tout d'abord il s'agit d'annoter manuellement les modèles de tâches, et ensuite de définir manuellement la structure de navigation. La structure de navigation est un graphe spécial, appelé graphe de dialogue. Les graphes de dialogue sont constitués de nœuds, pour la plupart interprétés comme des vues, et de transitions entre ces nœuds. Il existe différents types de nœuds qui distinguent les vues en termes de complexité, de hiérarchie et de modalité. Les transitions sont des relations dirigées d'un élément d'une vue vers une autre vue ou élément. Les transitions reflètent les aspects de navigation des interfaces utilisateur. Les éléments d'une vue sont en réalité les tâches du modèle de tâches. La définition de la structure de navigation revient donc à créer manuellement des vues avec un éditeur EMF / GMF et à affecter les tâches à ces vues.

Les auteurs ont implémenté un certain nombre d'outils pour assurer l'utilisation de leur méthode. Au départ ces outils étaient distincts et communiquaient à travers des langages propriétaires basés sur XML. Ces outils ont vu leur nombre et variété augmenter au fil du projet, ils ont alors été transférés dans l'environnement EMF d'Eclipse et la communication entre ces outils se fait à travers XMI.

Modèle de tâches et modèles de dialogue. Leur approche consiste plus précisément à demander au concepteur d'annoter les tâches de son modèle de tâches avec des marqueurs. Ces marqueurs permettent à un interprète de décider des tâches qui appartiennent à la même vue. Dans leur cas, l'interprète est le *dialog graph editor*. Les concepteurs peuvent soit joindre manuellement les informations supplémentaires au sein de l'éditeur de tâches, soit spécifier des profils généraux pour les opérateurs, qui sont utilisés pendant le processus de transformation. Toute tâche peut ainsi être marquée comme :

- ICV (Integrate Children into View) : une tâche et ses descendants de premier niveau sont attachées à la même vue.
- IN (Ignore Node) : la tâche et tous ses descendants sont ignorés au niveau du modèle de navigation.
- PUNM (Pick Up for Navigation Model) : les tâches abstraites sont incluses dans le modèle de navigation.
- VL (View List) : un regroupement explicite et un arrangement dans l'ordre des tâches enfants.

La partie haute de la Figure 44 montre un modèle de tâches CTT annoté. La partie basse montre un extrait du dialog graph (structure de navigation) editor qui est créé en se basant sur le modèle de tâches annoté. Cette partie basse montre trois vues simples avec une transition séquentielle entre elles. En effet, lors de la phase de transformation du modèle de tâches annoté en modèles de dialogue, la tâche abstraite est ignorée et ses enfants sont regroupés conformément à l'ordre « {0} {1,2} {3} » mentionné dans l'annotation. Par exemple si le concepteur avait activé ICV au lieu de VL, il obtiendrait toutes les tâches dans une seule vue.

Les profils de marqueurs sont des configurations prédéfinies qui peuvent être appliquées à tout ou uniquement aux nœuds sélectionnés de modèles de tâches ; en tant que tels ils peuvent être indépendants de la plateforme.

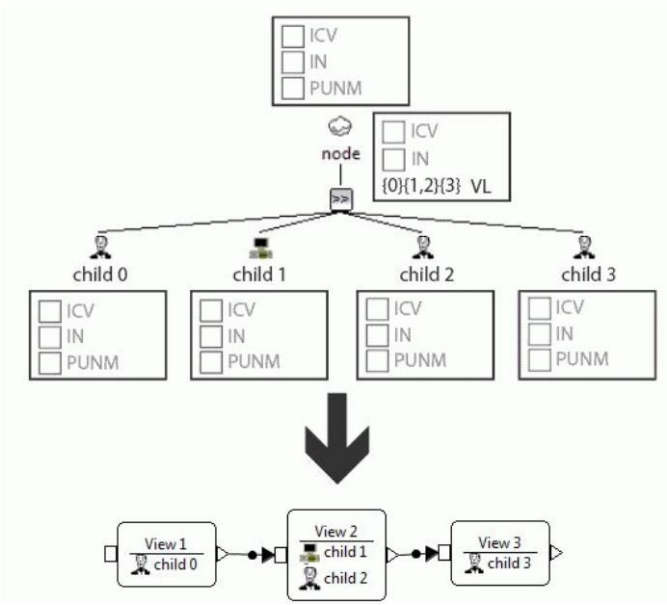


Figure 44 Transformation d'un modèle de tâches annoté

La génération de l'interface utilisateur. La combinaison du modèle de tâches et de la structure de navigation permet de générer un modèle d'interface abstrait. Ce modèle décrit l'architecture interne de chaque vue, c'est-à-dire les positions graphiques relatives aux tâches dans la vue. De plus, la classe des éléments d'interactions est assignée. Par exemple, une entrée 1 : n peut signifier (choisir dans la liste comme élément associé) ou une sélection hiérarchique peut signifier (arbre comme élément associé), ou un String (étiquette comme élément associé).

L'interface abstraite est transformée en interface concrète. Pour cela, la définition abstraite de l'architecture et les éléments d'interaction sont mappés à une architecture précise. Les classes d'objets d'interactions sont remplacées par des widgets implémentés. Ce modèle est finalement utilisé par les générateurs de code pour fournir une application ou une interface utilisateur final.

Les auteurs ont ainsi essayé de générer tout le processus (du passage du modèle de tâches vers le modèle d'interface concrète) mais ils ont constaté qu'une génération purement automatique ne produit pas des interfaces acceptables d'un point de vue de l'utilisabilité. La combinaison d'automatismes et de connaissances des concepteurs assistés par des outils de transformations donne des résultats plus acceptables.

1.3.5 Les travaux de Tran et al, 2009 - Generating user interface from task, user and domain models

Les travaux de (Tran et al. 2009) consistent à utiliser les modèles de tâches, de domaine et d'utilisateur pour générer des interfaces de base de données. La différence avec les travaux précédents est qu'ils combinent trois principaux modèles pour la génération de l'interface utilisateur et de codes pour les grandes fonctions d'une application de base de données comme *Display()*, *AddNew()*, *Update()*.

Les différents modèles servent un but précis à différentes étapes du processus de conception. Le *modèle de tâches* exprime les connaissances requises ou les procédures utilisées pour effectuer une tâche ; le *modèle de l'utilisateur* décrit les capacités de l'utilisateur et ses

préférences. Le *modèle de domaine* fournit les attributs des objets et les relations entre ces objets pour la création d'une interface utilisateur.

Le *modèle de tâches* est donc utilisé pour spécifier une interface utilisateur générique ; le *modèle de domaine* est utilisé pour spécifier les objets permettant d'interagir avec cette interface utilisateur. À ce niveau l'interface utilisateur est spécifiée avec plus de détails. Le *modèle de l'utilisateur* est utilisé pour influencer la conception et choisir parmi les solutions alternatives dans l'espace de conception.

La Figure 45 représente l'architecture de cette solution. Elle utilise les approches à base d'agents. Les agents utilisés sont : l'agent *model analyst*, l'agent *function analyst*, l'agent *UI creator* et l'agent *code generator*.

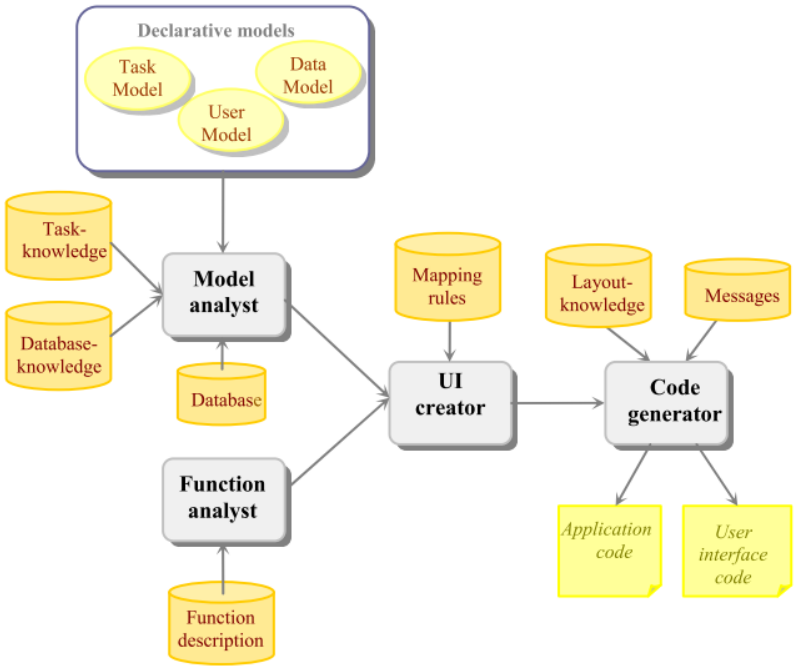


Figure 45 Les composants principaux de leur architecture de génération d'interface et de codes

L'agent *model analyst* est utilisé pour charger les modèles de tâches, de l'utilisateur et du domaine. Afin d'obtenir le comportement souhaité d'une tâche d'application de base de données, l'agent *function analyst* définit les fonctions de base de l'application en utilisant la base de description de la fonction (*Function description*). Ce sont par exemple Diplay(), AddNew() etc.

Une fois que les modèles d'entrée ont été chargés, le développeur détermine les tâches à partir desquelles l'interface utilisateur peut être générée en se basant sur le modèle de domaine (Figure 46). Ces tâches sont typiquement celles qui permettent de manipuler une base de données. Le développeur fait le lien entre les tâches spécifiées et les attributs des objets de domaine dans le modèle de domaine. Il fait également le lien entre les autres tâches et les fonctions définies.

Une fois que les tâches dans le modèle de tâches ont été liées aux attributs des objets de domaine dans le modèle de domaine par le *développeur* (Figure 46), des objets d'interaction concrète (CIO) sont créés sur la base des caractéristiques d'attributs et les relations entre les objets de domaine par l'agent *UI creator*. Ces caractéristiques sont par exemple, les types de données, longueur des données etc. Une fois les CIO créés, ils sont transformés en objets

d'interactions finales (FIO). Un FIO est décrit comme un mode de commande d'interface dans une plate-forme concrète.

Enfin, l'agent *code generator* utilise la base de connaissance *Layout - knowledge* pour générer le code d'interface utilisateur en se basant sur les FIO et utilise la base *Messages* pour générer le code d'application en s'appuyant sur les fonctions définies. Le code de l'application est généré pour effectuer les tâches liées aux fonctions qui sont définies par l'agent *function analyst*.

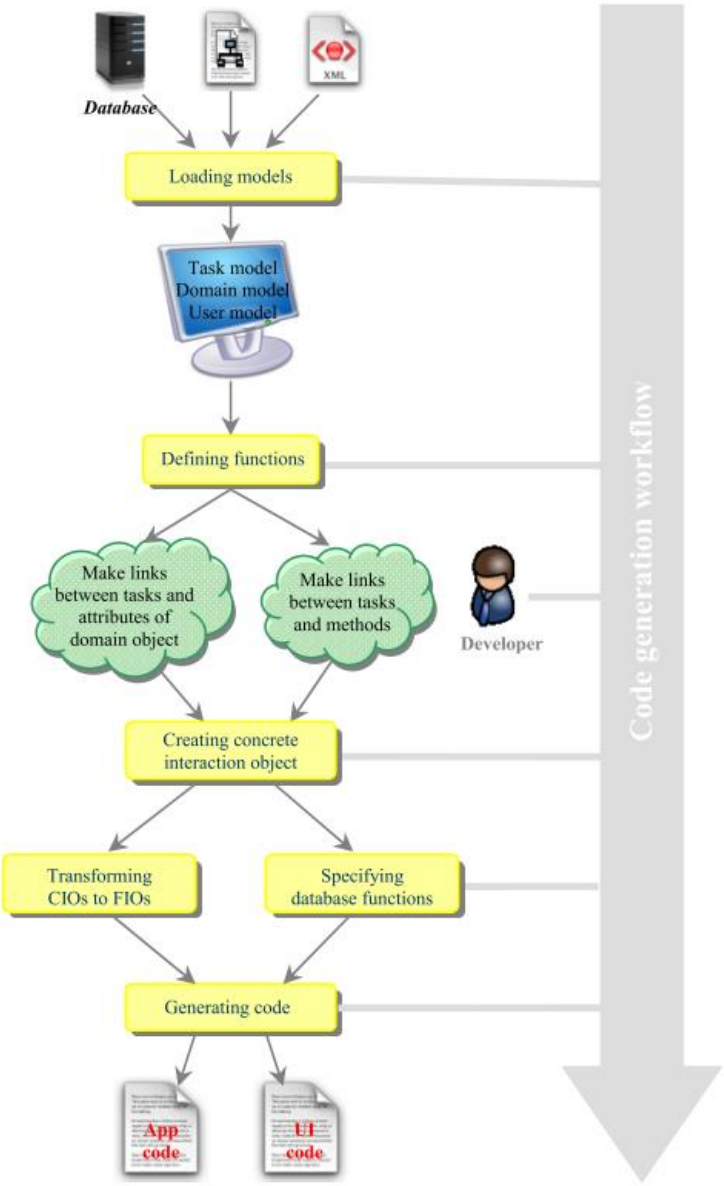


Figure 46 Flot de génération de code : interface utilisateur générée à partir du modèle de tâches, du modèle de domaine et du modèle de l'utilisateur.

La démarche proposée a été mise en œuvre à travers un Framework, puis appliqué à un projet concret, *Translogistic*. L'objectif de cette démarche est de générer automatiquement une IHM de base de données et du code en combinant le modèle de tâches, le modèle de domaine et modèle de l'utilisateur. L'intérêt est d'offrir en peu de temps, un environnement de développement efficace, à faible coût aux concepteurs.

1.3.6 Les travaux de Raneburger et al, 2012 – An Automated Layout Approach for Model-Driven WIMP-UI Generation

La motivation derrière les travaux de Raneburger et ses collègues (Raneburger, Popp, et Vanderdonckt 2012) est de résoudre le problème lié à la génération automatique des WIMP-UI (Window / Icon / Menu / Pointing Device User Interface). Les auteurs précisent que l'une des raisons principales pour lesquelles la génération d'interface n'est pas devenue très fréquente est que les interfaces utilisateur générées sont peu utilisables. Dans certains travaux, pour garantir la génération automatique d'interface, le concepteur compromet généralement l'utilisabilité, qui est le facteur le plus déterminant d'une interface utilisateur (Gerrir Meixner, Paterno', et Vanderdonckt; Gerrir Meixner, Paternò, et Vanderdonckt 2011). Ce problème d'utilisabilité est dû au fait que les exigences non-fonctionnelles, telles que la disposition ou le style ne sont pas suffisamment prises en compte lors du processus de génération. Une bonne architecture est cruciale pour aider les utilisateurs à trouver rapidement ce qu'ils cherchent et rend l'aspect visuel attrayant. Selon le guide de Microsoft sur l'expérience utilisateur, l'architecture est le dimensionnement, l'espacement, et le placement de contenu dans une fenêtre ou une page.

Les auteurs ont également remarqué que la plupart des approches basées sur les modèles pour la génération d'interface, utilisent des modèles de présentation ou d'architecture créés manuellement. Les démarches automatiques ne sont appliquées qu'à une partie d'architecture très limitée.

Pour rendre les interfaces générées plus utilisables, les auteurs proposent une approche de génération automatique d'architecture qui supporte la spécification explicite des paramètres d'architecture indépendamment de tout dispositif, ainsi que des règles de transformations réutilisables. Dans leur approche, les paramètres d'architecture manquants sont automatiquement complétés avec le « Layout Hints » (Indices d'architecture) en fonction des préférences de défilement. Leur objectif est d'améliorer l'interface utilisateur générée en considérant les "indices" et en appliquant des heuristiques, plutôt que de résoudre le problème de génération d'interface pour lequel ils pensent qu'il n'y aucune solution générique.

Les Layout Hints sont utilisés pour affecter un widget UI (Figure 47) à une certaine région de son conteneur parent. Ils fonctionnent avec le principe de diviser pour régner, ce qui signifie qu'ils séparent un conteneur en différentes régions et distribuent les widgets sur ces régions. Le raisonnement suivi est que s'il y a moins de widgets à placer, il y a moins d'options. Cela augmente les chances de créer l'option désirée, ou au moins une option qui est aussi proche de celle désirée.

Widget	**Layout Hint**
Home Button	alignment-y: top alignment-x: left
Back Button	alignment-y: top
Logout Button	alignment-y: top alignment-x: right
Next Button	alignment-y: bottom alignment-x: right

Figure 47 Exemple d'affectation de Layout Hints

Les Layout Hints permettent au concepteur de capturer les exigences non-fonctionnelles relatives à l'architecture qui ne peuvent être capturées dans les modèles de haut niveau. Les

Layout Hints demandent moins d'effort que la création manuelle d'un modèle d'architecture qui n'est parfois pas réutilisable.

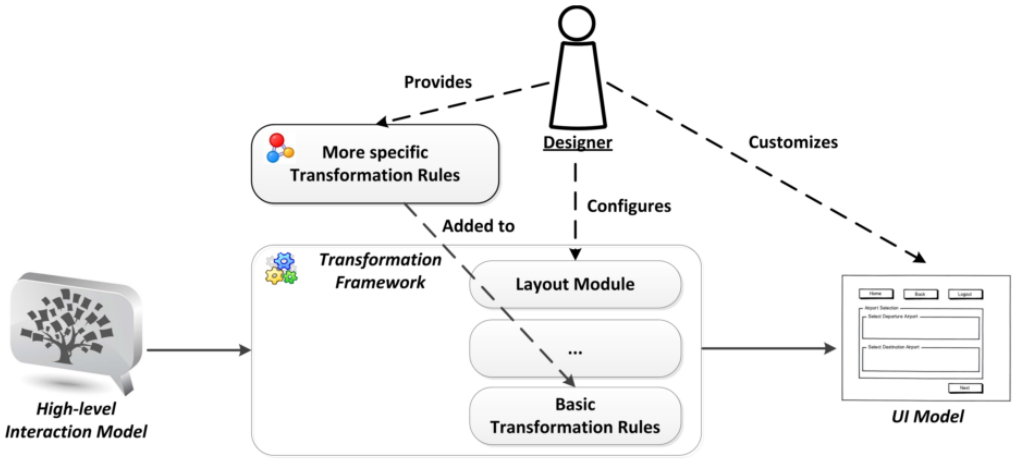


Figure 48 intégration du Layout Hints dans UCP:UI

L'approche de Layout Hints est intégrée dans une démarche plus globale, UCP : UI (Unified Communication Platform 4 User Interface generation framework) qui est un framework de génération d'interface illustré sur la Figure 48.

1.3.7 Les travaux de Ramon et al, 2015 – A layout inference algorithm for graphical user interface

La motivation derrière les travaux de (Ramón et al. 2015) est de franchir certaines des limites des approches de génération d'interface. Les auteurs précisent que les boites à outils d'interface graphique fournissent actuellement des gestionnaires de positionnement, qui ordonnent les widgets dans les vues en fonction de certaines contraintes qui caractérisent chaque type de gestionnaire de positionnement. Dans certains scénarios tels que la migration d'interface graphique et la génération automatique d'interface à partir des *wireframes* (le *wireframe* en web design consiste à réaliser un schéma définissant les zones d'un site Web, ou d'une page Web. Il peut être réalisé par une personne non technique), l'architecture des vues est implicitement exprimée par l'utilisation de coordonnées. Dans ces cas, il est souhaitable de représenter l'architecture explicite en termes de gestionnaires de postionnement.

La méthode qu'ils proposent est de représenter une interface graphique à base de coordonnées en un ensemble de gestionnaires de positionnement afin de fournir différentes solutions alternatives pour une vue donnée, puis de choisir la meilleure alternative. Cette méthode est basée sur un processus d'inférence de positionnement et garantit l'indépendance des gestionnaires de positionnement spécifiques et la capacité de générer plusieurs solutions alternatives.

Le processus proposé se déroule en deux phases. Tout d'abord, le système de positionnement à base de coordonnées est remplacé par un système de positionnement basé sur les graphes orientés et les relations Allen (un calcul pour les raisonnements spatio-temporels créé par James F. Allen en 1983. Ce calcul définit les relations possibles entre des intervalles de temps et propose une table de composition). Ensuite, un algorithme exploratoire basé sur le filtrage par motif et la réécriture de graphes est appliqué pour obtenir différentes solutions de positionnement. Les auteurs évaluent l'algorithme à travers des études de cas liées à la génération automatique d'interfaces web à partir de wireframes. Cette approche permet de

prouver que la représentation du gestionnaire de positionnement obtenue à partir des interfaces graphiques à base de coordonnées peut être utilisée pour générer des interfaces.

1.3.8 Synthèse de l'état de l'art

Les travaux présentés dans cette section s'intéressent à la génération d'interface. Certain travaux se focalisent sur la génération de la présentation (Paterno, Breedvelt-schouten M., et Koning 1999), d'autres sur la génération du dialogue (Andreas Wolff et Forbrig 2009) et d'autres encore combinent la génération du dialogue et de la présentation (TERESA, Dygimes). Le Tableau 2 présente une synthèse des démarches existantes.

Démarches existantes	Paterno et al, 1999	Berti et al. 2003 TERESA	Luyten 2004 DYGIMES	Wolf et Forbrig, 2009	Tran et al, 2009	Raneburger et al, 2012	Ramon et al, 2015
Modèles générés	Présentation	Présentation et dialogue inter-fenêtre	Présentation et dialogue inter-fenêtre	Dialogue et interface abstraite	Interface utilisateur et codes pour fonctions	Architecture (Présentation)	Architecture (Présentation)
Techniques et/ou outils utilisés	Algorithme de calcul des ETS, PSs	Algorithme de calcul des ETS, PSs	Algorithme de calcul des ETS, PSs	Marquage du modèle de tâches. Utilisation du dialog graph editor	Technique à base d'agents	Configuration d'un modèle de Layout par le concepteur	Inférence
Avantages	Démarche outillée	Possibilité d'appliquer des heuristiques. Démarche outillée	Démarche outillée	Visualisation du dialogue avec dialog graph editor	Génération automatique d'interface et de codes	Permet d'avoir une interface plus utilisable. Règles de transformations réutilisables	Garantit l'indépendance des gestionnaires d'architecture spécifiques. Capacité de générer plusieurs solutions alternatives
Inconvénients	Seulement utilisable quand le modèle de tâches est décrit avec CTT. Les dialogues intra-widget et intra-fenêtre ne sont pas décrits	Seulement utilisable quand le modèle de tâches est décrit avec CTT. Les dialogues intra-widget et intra-fenêtre ne sont pas décrits. Nombre limité d'heuristiques	Seulement utilisable quand le modèle de tâches est décrit avec CTT. Définition de lien non bijectif entre le modèle de dialogue et le modèle de tâches. Les dialogues intra-widget et intra-fenêtre ne sont pas décrits	Processus de marquage peut être fastidieux Les dialogues intra-widget et intra-fenêtre ne sont pas décrits	Spécifique au domaine de base de données. Peut nécessiter d'importantes modifications dans les bases et les agents pour être appliqué à d'autres domaines	Concerne seulement la partie statique de l'interface	Concerne seulement la partie statique de l'interface

Tableau 2 Synthèse des démarches existantes

Pour les travaux de (TERESA, Dygimes), le passage du modèle de tâches à l'un ou l'autre des modèles commence par le calcul des ETS (les ensembles de tâches activables dans un même laps de temps). Les travaux de (Andreas Wolff et Forbrig 2009) utilisent un type particulier d'annotations du modèle de tâches. (Tran et al. 2009) quant à eux, utilisent une technique à base d'agents pour déduire l'interface à partir de différents modèles sans passer par une génération de modèle de dialogue et/ou de présentation.

Dans TERESA et (Paterno, Breedvelt-schouten M., et Koning 1999), d'une part, les auteurs partent du principe que tous les ETS doivent être présentés dans une fenêtre. Les ETS ne doivent pas contenir l'opérateur *enabling* comme pour les travaux de (Paterno, Breedvelt-schouten M., et Koning 1999). Cependant, il est possible d'utiliser des heuristiques supplémentaires (Andreas Wolff et Forbrig 2009). D'autre part, la façon d'obtenir les PSs (ensembles de présentation) fait penser que le dialogue à l'intérieur de chacune d'entre elles n'est pas décrit (Caffiau 2010). En effet, chaque PSs est une boîte de dialogue dans laquelle toutes les tâches sont réalisables en parallèle.

Dygimes (Luyten 2004) fournit un processus simple et clair pour générer automatiquement des interfaces utilisateur multi-dispositif et multi-contexte. La conception de l'interface utilisateur commence par le modèle de tâches en tant que modèle central. Une fois que le

modèle de tâches est créé avec la notation appropriée (CTT), d'autres modèles (modèle de présentation et modèle de dialogue) sont générés à partir de ce modèle de tâches. En utilisant plusieurs algorithmes, la cohérence entre les différents modèles est assurée, et le concepteur de l'interface utilisateur obtient un système interactif correspondant au modèle de tâches défini. Comme dans TERESA, Dygimes Framework, associe à chaque tâche (non réalisée entièrement par un utilisateur) une transition dans le dialogue de l'application, définissant ainsi un lien entre les deux modèles (modèle de dialogue et modèle de tâches). Cependant, ce lien entre les transitions et les tâches n'est pas un lien bijectif, par exemple, le dialogue doit prendre en compte les transitions de retour en arrière alors que le modèle de tâches ne les détaille pas nécessairement (Caffiau 2010).

L'approche proposée dans (Andreas Wolff et Forbrig 2009) peut être considérée comme une stratégie alternative pour les concepteurs, si le processus de génération ne donne pas les résultats escomptés. Il peut dépendre du domaine d'application et notamment du nombre de tâches dans le modèle initial. Cette approche peut être utilisée même lorsque le nombre de tâches est important, car les tâches ne sont pas toujours exécutées de façon simultanée. Cependant le processus de marquage des tâches peut être fastidieux.

L'approche de (Tran et al. 2009) n'utilise pas que le modèle de tâches, il utilise également le modèle utilisateur et le modèle de domaine. La combinaison de modèles est un concept important dans la génération d'interface utilisateur puisque les différents modèles décrivent différents aspects de l'interface utilisateur. Par exemple, une interface utilisateur générée à partir d'un modèle de tâches devrait être un moyen sur lequel l'utilisateur peut communiquer avec le système pour accomplir cette tâche. L'interface utilisateur générée à partir d'un modèle de l'utilisateur est prévue pour supporter les utilisateurs en fonction de leurs caractéristiques. L'interface utilisateur générée à partir de plusieurs modèles différents comporte de nombreux aspects essentiels. Cependant, cette approche s'appuie sur une technique à base d'agents avec des règles prédéfinies pour le domaine des bases de données. L'application de cette approche à d'autres domaines n'est pas précisée et peut nécessiter d'importantes modifications dans les bases et les agents.

Pour améliorer les interfaces utilisateur générées, les auteurs du (Raneburger, Popp, et Vanderdonckt 2012) ont proposé une approche de génération automatique de positionnement qui supporte la spécification explicite des paramètres de positionnement indépendamment de tout dispositif ainsi que des règles de transformations réutilisables. Ils utilisent des « Layout Hints » qui permettent au concepteur de capturer les exigences non-fonctionnelles relatives au positionnement qui ne peuvent être capturées dans les modèles de haut niveau. Cette technique demande moins d'effort que la création manuelle d'un modèle de positionnement qui n'est parfois pas réutilisable.

Dans (Ramón et al. 2015), les auteurs proposent un processus d'inférence de positionnement pour la génération d'interface. Dans leurs travaux, ils ne s'intéressent qu'à la génération de la partie statique des interfaces, en particulier le positionnement des widgets dans les vues de l'application. Leurs travaux s'appliquent au domaine du web.

1.4 Verrous identifiés par l'état de l'art sur la génération d'interface à partir des modèles de tâches

La génération des applications interactives à partir du modèle de tâches est un réel défi car cela revient à générer les modèles de présentation et de dialogue. Certains travaux proposent

des démarches de génération automatique ou semi-automatique permettant d'obtenir toute ou partie de l'application. D'autres complètent les modèles de tâches par des données relatives aux modèles de présentation et de dialogue pour faciliter la génération de l'application.

La génération de la présentation à partir du modèle de tâches a fait l'objet de plusieurs travaux de recherche. La plupart des travaux focalisés sur cette génération passe par l'ajout d'items au modèle de tâches, pour faciliter l'obtention de la présentation. Cependant, cette solution présente le désavantage (Caffiau 2010) d'insérer dans le modèle de tâches des données qui n'appartiennent pas au niveau d'abstraction des modèles de tâches. Générer la présentation sans ajouter d'items au modèle de tâches, constitue le *premier défi* que nous avons identifié.

La génération du dialogue à partir du modèle de tâches s'appuie sur le principe d'animation du modèle de tâches utilisé par les outils de simulation de tâches. Partant de ce principe, le dialogue semble être simple à obtenir. Cependant, l'étude réalisée par les travaux de (Caffiau 2010) montre que cela n'est pas si facile. En effet, malgré la complexité de certains modèles de tâches, ces derniers ne permettent pas généralement de dériver le la totalité du dialogue. Celle-ci nécessite la mise en œuvre de transformations. Au cours de ces transformations, il n'est pas évident de conserver le lien entre le modèle de tâches et le modèle de dialogue. L'obtention du modèle dialogue sans passer par des transformations, tout en conservant en permanence le lien avec le modèle de tâches, constitue le *deuxième défi* que nous avons identifié.

Dans la plupart des travaux permettant la génération à partir des modèles de tâches, la déduction des relations entre les tâches ne concerne que le dialogue inter-fenêtres, à l'exclusion des dialogues intra-widget et intra-fenêtre. Considérer les trois types de dialogue pour une génération complète des interfaces de contrôle, constitue le *troisième défi* que nous avons identifié.

Par ailleurs, les deux types de dialogues (dialogue intra-widget, dialogue intra-fenêtre) manquants peuvent être déduites des spécifications fonctionnelles du système à concevoir. Cependant, il faut tout d'abord apporter une solution aux verrous liés à l'obtention des spécifications fonctionnelles des systèmes complexes avant de pouvoir les utiliser dans la génération des systèmes interactifs.

Dans la prochaine section de ce chapitre, nous définissons des techniques qui semblent être une solution intéressante pour faciliter la spécification fonctionnelle des systèmes complexes.

2 Faciliter la spécification fonctionnelle des systèmes complexes

Mettez-vous à la place d'un concepteur expert en charge de la spécification d'un système et imaginez que vous ayez un collaborateur qui vous assiste dans votre tâche de spécification. Lorsque vous lui demandez de faire quelque chose que vous lui aviez montré, il le fait exactement comme vous le souhaitez. Il est même en mesure de vous proposer en un rien de temps, plusieurs autres façons d'effectuer la même tâche. Pour vous permettre de vérifier ses propositions, il est capable de vous montrer comment les spécifications qu'il vient de définir fonctionneraient en situation réelle. Si vous détectez des erreurs dans son travail, il suffit juste de lui faire part des corrections pour qu'il vous propose de nouveaux résultats. Une fois les erreurs corrigées, vous pouvez à nouveau tester les spécifications pour les vérifier et les valider. Ainsi, il vous réduit la charge de travail et vous aide à détecter plus rapidement les

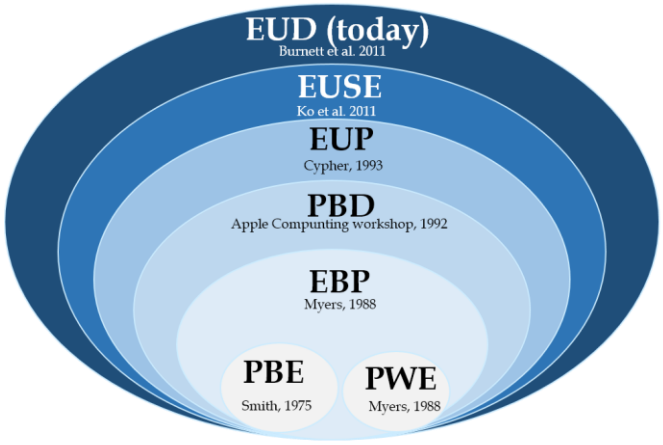
incohérences. Vous n'avez pas besoin d'apprendre de nouvelles techniques, ni de nouveaux langages pour le comprendre. Ne seriez-vous pas satisfait du travail de ce collaborateur qui vous fournit des résultats rapides et concrets afin de vous faciliter la spécification des systèmes complexes ?

Il existe depuis quelques années des systèmes qui fonctionnent de la façon dont travaille ce collaborateur. Ces systèmes sont conçus pour permettre aux utilisateurs finaux (non-programmeurs) de créer, modifier et étendre une application à leur propre besoin. L'approche utilisée dans ces systèmes est celle d'End User Development *(EUD)*. Cette approche est étudiée dans cette section pour déterminer ses potentiels apports et son utilisabilité dans la résolution de nos verrous.

Dans cette section, nous définissons les concepts de l'**EUD.** Nous nous intéressons aux techniques d'EUD pour en expliquer le fonctionnement, puis nous nous attardons sur quelques systèmes utilisant les techniques définies afin de les illustrer.

2.1 Introduction sur l'EUD

Le développement par utilisateur final permet de rendre la programmation accessible aux non-programmeurs. Les premières recherches ont commencé en 1975 par les travaux de (Smith 1977) avec le système Pygmalion sur le thème de Programming by Example (PBE). Par la suite, certains travaux ont proposé d'autres formalisations de ce thème en fonction de la prise en compte ou non des aspects qu'ils jugent importants dans les systèmes qu'ils conçoivent. La Figure 49 rend compte de ces différentes évolutions allant de PBE jusqu'au EUD. La définition des acronymes est renseignée dans le tableau présenté sur la Figure 49b.



(a)

Abréviations	Définition du terme
EUD	End User Development
EUSE	End User Software Engineering
EUP	End User Programming
PBD	Progamming By Demonstration
EBP	Example Based Programming
PBE	Programming By Example
PWE	Proframming With Example

(b)

Figure 49 Évolution des différents thèmes : du EBP au EUD

Selon (Burnett et Scaffidi 2011), l'EUD est un ensemble de méthodes, techniques et outils permettant à un non-informaticien de créer, de modifier et d'étendre un logiciel . Concrètement, l'EUD permet aux utilisateurs finaux de concevoir ou d'adapter l'interface utilisateur et les fonctionnalités du logiciel à leurs besoins. Cela est utile parce que les utilisateurs finaux connaissent leur propre contexte et leurs besoins mieux que quiconque (Burnett et Scaffidi 2011). L'EUD recoupe l'ensemble des thèmes présentés sur la Figure 49. La dernière appellation de ce thème, l'EUSE (End User Software Engineering) a été formalisée pour tenir compte de la qualité des programmes fabriqués.

L'*EUSE* a émergé il y a environ une décennie. Ce paradigme met l'accent sur la qualité du logiciel que les utilisateurs finaux créent, modifient ou étendent. Ce domaine est né du fait que de nombreux éléments de preuves confirment que les programmes créés par les utilisateurs finaux sont remplis d'erreurs, dont la correction est coûteuse (Cao et al. 2010). Ainsi les travaux dans ce domaine se concentrent sur les méthodes, techniques et outils qui favorisent la qualité d'un logiciel (Ko et al. 2011). Les techniques du EUD permettent aux utilisateurs finaux non-programmeurs de créer, parfaire, partager et de changer leur façon de travailler pour obtenir des logiciels de qualité. Ces qualités concernent la conformité de l'*aspect fonctionnel* du logiciel, la *fiabilité*, la *réutilisabilité*, la *facilité de maintenance*, la *performance*, la *protection*, la *sécurité* etc. (Burnett et Scaffidi 2011) et sont appliquées à l'ensemble du cycle de vie d'EUD (Figure 50). La *fiabilité* concerne la détection et la correction d'erreurs dans le programme conçu par des utilisateurs finaux (non-programmeurs).

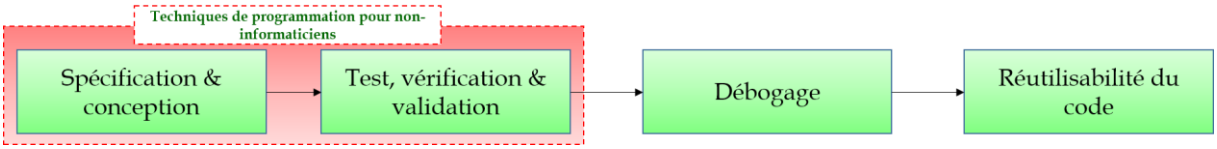


Figure 50 Cycle de vie d'EUD

Le cycle de vie de l'EUD vient compléter les outils de création de programmes qui s'appuient sur les techniques de l'EUP, pour appliquer les qualités exigées dans une conception classique d'un logiciel par les développeurs professionnels, aux programmes créés par des utilisateurs finaux. Ce cycle de vie comprend quatre grandes parties : la *spécification et la conception*, la *vérification et la validation*, le *débogage* et la *réutilisabilité* du code créé.

Dans ce mémoire nous utiliserons la terminologie **EBP** (Example Based Programming) car l'utilisateur que nous considérons n'est pas l'utilisateur final mais plutôt l'utilisateur expert (expert métier) qui est le concepteur de système de contrôle-commande.

2.2 Le cycle de vie et les techniques du EUD

2.2.1 La spécification et la conception

La spécification des exigences et la conception sont fortement liées car les exigences définissent ce qu'un programme doit faire (*what should my program do ?*) et la conception définit comment le programme atteint les objectifs décrits dans les spécifications (*How should a program work ?*).

2.2.1.1 La spécification des exigences

Généralement, dans une conception classique réalisée par des développeurs professionnels, la description des spécifications est une étape importante. Ces exigences sont décrites dans des langages et formalismes laborieux (voir section I.2.) pour respecter certaines propriétés décrites dans la section I.2.1 afin d'assurer des logiciels de meilleure qualité. Dans le cas de l'EUD, on suppose que les utilisateurs connaissent bien leur domaine et donc, la plupart du temps, les exigences. Cependant, comme les développeurs professionnels qui utilisent les méthodes agiles, les utilisateurs finaux peuvent ne pas connaître à l'avance les exigences (Segal 2007) ; mais pendant que les développeurs professionnels mettent en œuvre des techniques pour recueillir les exigences, les utilisateurs non-professionnels passeront directement à la création du code. Le processus de raffinement d'obtention des exigences a été comparé par (Lieberman et al. 2006) à une forme de programmation très agile, en raison de la nature hautement itérative de ce dernier. Les non-programmeurs utilisant un système d'EUD programment généralement pour eux-mêmes ou pour des collègues ou amis. En EUD, le défi

de la spécification des exigences est donc de comprendre trois éléments importants : le contexte, les besoins et les priorités des personnes et organisations à qui est destiné le programme conçu (Ko et al. 2011). L'assurance de la compréhension de ces éléments est primordiale pour une bonne exploitation d'un système intégrant les techniques d'EUD. Dans la plupart des cas, ces éléments sont facilement compréhensibles par des non-programmeurs (car dans une conception classique ce sont eux qui définissent ces exigences) et les spécifications peuvent facilement évoluer.

Dans la littérature, quelques systèmes permettent de définir les exigences pour la création de programmes par des non-informaticiens. Deux exemples de systèmes qui tiennent compte de la définition des exigences sont : Whyline (Ko, Myers, et Aung 2004) et le paradigme de Spreadsheet (Abraham et Erwig 2007a). Whyline permet aux utilisateurs de se demander le « pourquoi » de la création de leurs programmes. Les réponses à ces questions leur permettent d'apprendre implicitement ce qu'ils attendent de leurs programmes. Le paradigme de feuille de calcul quant à lui, permet aux utilisateurs d'examiner des valeurs incorrectes dans la feuille de calcul produite.

2.2.1.2 La conception

La conception consiste à traduire les exigences en un programme qui fonctionne. Pour les programmeurs non-informaticiens, les activités de description des exigences et de conception sont rarement distinctes. En effet, les systèmes d'EUD visent à soutenir un prototypage évolutif et exploratoire. Par exemple, le système DENIM (Lin et al. 2001) offre un prototypage évolutif et exploratoire. Ce système permet aux utilisateurs de laisser des parties de l'interface dans un état brut et ambigu jusqu'à mieux comprendre son utilisation. Le processus utilisé dans ce système est parfois assisté par une conception critique. La conception critique est une caractéristique introduite dans un système d'EUD qui permet de revoir la conception d'un utilisateur et donner des suggestions d'amélioration (Burnett et Scaffidi 2011). C'est le cas de Janus (Fischer et Girgensohn 1990), un outil de conception par utilisateur final des plans pour une maison. Cet outil permet d'identifier les combinaisons optimales des objets placés, dans les plans, par les utilisateurs finaux et renvoie des suggestions avec justifications pour corriger la disposition des objets.

Laisser les non-programmeurs travailler comme ils le font habituellement peut être un moyen d'appliquer une bonne conduite dans la conception de leur programme (Ko et al. 2011).

2.2.1.3 Mise en œuvre de la spécification et de la conception dans un système d'EUD

Généralement dans les systèmes de EUD, la spécification et la conception sont mises en œuvre suivant les techniques d'enregistrement et de généralisation.

2.2.1.3.1 L'enregistrement

Le mode *enregistrement* permet à un système intégrant les techniques de l'EUD d'enregistrer des actions utilisateur afin d'en conserver une trace. Le système peut enregistrer soit les actions utilisateur, soit la réaction du système aux actions utilisateur, ou même les deux. Les données obtenues au cours de la phase d'enregistrement dépendent du type de système manipulé par l'utilisateur. Certains systèmes de robotique permettent d'enregistrer l'utilisateur manipulant les objets physiques dans un environnement bien défini. Les informations importantes comme les contraintes rigides sont extraites des données enregistrées et chargées dans une interface graphique (Mollard et al. 2015), pour leur manipulation. Dans la domotique, les systèmes sont dotés d'une interface graphique permettant à l'utilisateur d'exprimer ses besoins de façon

textuelle en utilisant des informations prédéfinies sur l'interface. Pendant qu'il manipule ces informations le système enregistre un programme pouvant être exécuté. Pour d'autres types de systèmes, les enregistrements se font sur une interface graphique assez familière à l'utilisateur pouvant lui permettre d'exprimer de façon singulière ce qu'il veut que son système fasse. Dans ses travaux, (Sanou 2008) détaille cinq différents niveaux d'enregistrement : *enregistrement des actions les plus élémentaires*, *enregistrement de l'aspect lexical de l'interaction*, *enregistrement de l'aspect syntaxique de l'interaction*, *enregistrement de la sémantique*, *enregistrement des actions pragmatiques*. Nous présentons dans le Tableau 3, un récapitulatif des différents niveaux d'enregistrement.

Niveaux d'enregistrement	Caractéristiques	Exemples de système	Avantages/ Inconvénients
1. Actions les plus élémentaires	Enregistrement des évènements de bas niveau. Les enregistreurs sont insérés dans les objets de la boite à outils d'interaction du système	Enregistreur de macros, Jacareto[16]	- La situation initiale joue un grand rôle - Toute différence dans la configuration initiale de l'écran aboutit à une exécution erronée de la macro - Nécessiterait un enregistrement pour chaque situation
2. Aspect lexical de l'interaction	L'enregistrement des actions élémentaires se fait du point de vue de l'utilisateur	Eager (Cypher 1991)	- Anticipe les actions de l'utilisateur pour lui proposer une solution. - Les conditions de réutilisation du programme ne sont pas communiquées.
3. Aspect syntaxique de l'interaction	Enregistrement du dialogue entre l'utilisateur et le système	Like (Girard 1992)	Permet une meilleure interprétation des actions de l'utilisateur
4. Sémantique	Enregistrement de la fonction du système invoquée par l'utilisateur	EBP (Potier 1995)	Permet de créer de véritables programmes paramétrés, échangeables d'un système à un autre
5. Actions pragmatiques	Enregistrement, uniquement, des actions du système qui ont un rôle dans le but final de l'utilisateur	Systèmes paramétriques (Texier 2000)	Permet un rejeu après d'éventuelles modifications de paramètres

Tableau 3. Récapitulatif des différents niveaux d'enregistrement

Généralement, le type d'enregistrement est choisi en fonction de l'objectif attendu du système d'EUD. En effet, l'objectif de l'utilisation d'EUD peut être par exemple de construire des programmes réutilisables ou à l'opposé d'enregistrer des actions qui seront utilisées à l'identique, sans aucune modification dans l'application. Pour construire des programmes génériques réutilisables, l'enregistrement est complété par une phase de généralisation.

2.2.1.3.2 La généralisation

La généralisation dépend fortement de la structure de l'enregistrement et permet d'appliquer une même procédure à de nouveaux exemples. La différence entre les systèmes d'EUD réside dans la technique de généralisation utilisée par ces derniers. Certains systèmes demandent à l'utilisateur d'indiquer explicitement les propriétés des exemples qui doivent être généralisés. D'autres systèmes contiennent des algorithmes qui envoient des propositions à l'utilisateur afin de l'impliquer dans la généralisation. D'autres encore s'appuient sur des techniques d'inférences qui utilisent des heuristiques pour déduire la généralisation de l'exemple. La Figure 51 présente les types et sous-types de généralisation que l'on peut rencontrer.

[16] http://jacareto.sourceforge.net/wiki/index.php/Main_Page

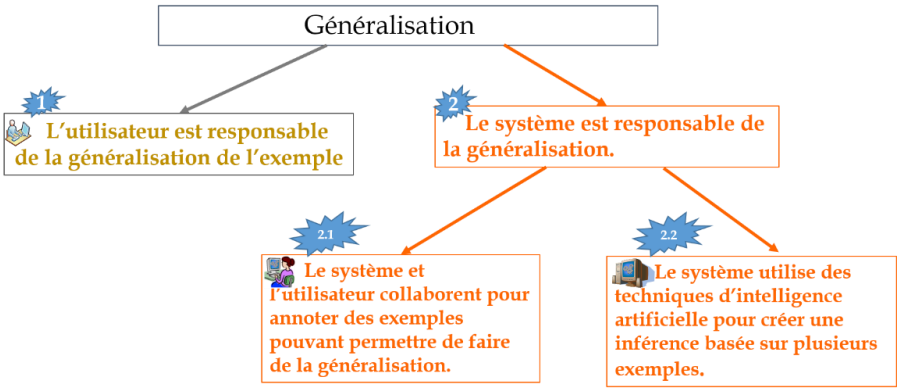


Figure 51 Les différents types de généralisation

Pour le premier type, *l'utilisateur est responsable de la généralisation de l'exemple*. ToonTalk (Kahn 2000) et Topaz (Brad A. Myers 1998) sont deux exemples de systèmes qui intègrent ce type de généralisation. Dans ToonTalk, l'utilisateur généralise les actions enregistrées en supprimant explicitement les détails (valeurs). Dans Topaz l'utilisateur doit explicitement généraliser les paramètres des opérations en utilisant des boites de dialogues.

Le deuxième type de généralisation indique que c'est *le système qui est responsable de la généralisation des exemples*. Pour ce type de généralisation, on distingue deux cas, selon que l'utilisateur participe ou non à la généralisation.

Dans le premier cas, *le système tient compte de l'utilisateur* dans le processus de généralisation. Peridot (Bras A. Myers et Buxton 1986) est l'un des premiers systèmes d'EUD doté de ce type de généralisation. Il utilise beaucoup d'heuristiques, des règles basées sur des inférences d'un simple exemple et, pour chaque règle détectée, il demande à l'utilisateur de confirmer l'inférence. Peridot utilise des questions-réponses pour confirmer chaque inférence, ce qui se révèle être problématique parce que les utilisateurs ont tendance à répondre simplement « *oui* » à toutes les questions, apparemment en supposant que l'ordinateur savait ce qu'il faisait. Le système Pursuit (Modugno et Myers 1997) offre une représentation graphique qui permet à l'utilisateur de confirmer ou de corriger les inférences proposées. Ces inférences seront utilisées plus tard pour l'édition des programmes. SmallStar (Halbert 1993) fournit une représentation textuelle du programme créé à partir d'un exemple, et l'utilisateur doit modifier explicitement le programme pour généraliser les paramètres et ajouter des structures de contrôle. Avec ce type de généralisation, il est facile pour l'utilisateur de comprendre comment le système procède pour généraliser l'exemple. Cependant, les propositions envoyées par le système à l'utilisateur sont basées sur le comportement d'un seul exemple ce qui limite l'utilisation du code généré issu de la généralisation à quelques cas. En effet, dans certains cas, le code généré doit être modifié pour être utilisé.

Dans le deuxième cas, les systèmes ne fournissent aucune rétroaction sur les programmes inférés. Par exemple, le système Gamut (McDaniel et Myers 1999) infère les comportements de plusieurs exemples. Il commence la construction d'un nouveau comportement de la même manière que de simples exemples. Chaque fois que l'utilisateur affine le comportement en ajoutant un nouvel exemple, Gamut utilise des métriques pour comparer le nouvel exemple avec le comportement existant pour déduire la façon dont le comportement doit être changé. Ce système permet aux utilisateurs de démontrer de nouveaux exemples à tout moment, pour modifier les programmes généralisés existants, il n'est donc pas nécessaire d'éditer le code généralisé. L'avantage principal de l'utilisation des techniques d'inférence plus sophistiquées

est que le système peut déduire des comportements plus complexes à partir des actions de l'utilisateur. Cependant, les systèmes qui intègrent ce type de généralisation sont beaucoup plus difficiles à développer et ils peuvent ne pas inférer correctement, ce qui peut conduire à des erreurs dans le code généré.

2.2.2 Test, vérification et validation

Cette phase permet de s'assurer de l'exactitude du programme créé par les non-programmeurs. Plusieurs techniques sont utilisées pour s'assurer que le programme répond à l'objectif attendu. Parmi ces techniques, nous pouvons citer : les *tests*, la *vérification des spécifications*, la *vérification de la cohérence*, la *visualisation* etc.

Les *tests* permettent d'aider les non-programmeurs à détecter des erreurs dans leurs programmes. En EUD, les tests systématiques permettent de diminuer la probabilité de manquer d'importantes erreurs (Gregg Rothermel et al. 2001). Ce type de test s'appuie sur un plan qui définit exactement ce qui doit être testé et précise si le nombre de tests est suffisant.

La *vérification des spécifications* permet de confronter les valeurs renvoyées, par le programme créé par les non-informaticiens, aux spécifications (2.2.1). Ainsi, les spécifications servent d'exemples parfaits pour la vérification de la conformité du programme créé.

La *vérification de la cohérence* consiste à intégrer dans le système d'EUD des heuristiques sur la cohérence interne du programme. Lors de la phase de vérification, le système utilisera ces heuristiques pour vérifier la conformité du programme créé par les non-informaticiens. Les approches qui appliquent ce type de vérification, utilisent l'analyse statistique et des techniques pour attirer l'attention des programmeurs non-professionnels sur les valeurs potentiellement problématiques (Ko et al. 2011).

La *visualisation* est une forme de vérification qui permet aux programmeurs non-informaticiens de visualiser le comportement du programme créé pour vérifier sa conformité. Pour cela ils s'appuient sur leur expertise du domaine ou la connaissance qu'ils ont du fonctionnement du programme.

L'objectif principal de la phase de tests, vérification et/ou validation est de détecter les erreurs qui se trouvent dans les programmes créés par les programmeurs non-professionnels.

2.2.2.1 Mise en œuvre du test, vérification et validation : le rejeu

La phase de rejeu permet aux systèmes de rejouer toutes les alternatives proposées par la généralisation et offre ainsi à l'utilisateur la possibilité de vérifier ces possibilités. Un système d'EUD est capable de rejouer des séquences d'actions répétitives et contenant des boucles (Ruvini et Dony 2001). Cependant l'état dans lequel se trouve le système peut influencer le rejeu. Un comportement illustré lors de l'enregistrement des exemples ne sera peut-être pas le même lors du rejeu dépendant du contexte (état du système).

L'utilisateur peut avoir lors du rejeu, deux objectifs : il peut, soit chercher à vérifier ses interactions élémentaires mises en évidence par le rejeu, soit chercher à vérifier uniquement le résultat final. Les systèmes intégrant les techniques d'EUD offrent la possibilité à l'utilisateur d'atteindre l'un ou l'autre de ces objectifs ou même les deux.

Le rejeu peut nécessiter l'intervention de l'utilisateur pour orienter le système et pour confirmer ou non les résultats de la généralisation.

2.2.3 Le débogage

Le débogage permet de trouver et de corriger les erreurs détectées lors de la phase de tests et de vérification. La plupart des non-programmeurs ignorent comment leur programme est exécuté et ont donc souvent des difficultés pour expliquer les erreurs détectées (Ko, Myers, et Aung 2004). Avec les systèmes d'EUD, les non-programmeurs privilégient l'atteinte de leurs objectifs sur la fiabilité du programme conçu. Les solutions de débogages intégrées à ces systèmes sont souvent « rapides et grossières » (Ko et al. 2011). L'une de ces solutions est la modification du code jusqu'à ce qu'il fonctionne correctement. De telles stratégies révèlent souvent des erreurs supplémentaires (Ko et Myers 2003; Beckwith et al. 2005). Cependant, certaines approches de débogage adaptent des techniques utilisées par les développeurs professionnels et les intègrent aux systèmes d'EUD. Dans ces systèmes, en plus de l'affichage des valeurs des variables que le programme exécute, les non-programmeurs peuvent parcourir les instructions, une par une, en cherchant les opérations incorrectes (Leshed et al. 2008). Une autre technique permet aux non-programmeurs d'insérer une condition dans le code créé pour trouver où se situe l'erreur dans le code (Scaffidi 2010). Dans Whyline, par exemple, le processus de débogage commence quand l'utilisateur peut cliquer sur un bouton « *Why* » après avoir observé un comportement inattendu dans l'exécution du programme créé. Ce click affiche un menu avec les questions « Pourquoi » et « Pourquoi pas » qui sont organisées suivant la structure des objets 3D visibles manipulés par le programme. Une fois que l'utilisateur sélectionne une question, le système analyse l'historique de l'exécution du programme et génère une réponse expliquant l'erreur produite par l'affichage des évènements lors de l'exécution. Dans GoalDebug (Abraham et Erwig 2007b), un débogueur semi-automatique utilisé dans les feuilles de calcul, l'utilisateur peut sélectionner une valeur erronée, donner une valeur attendue et obtenir un ensemble de changements apportés aux formules de la feuille de calcul. Les changements suggérés sont générés par un système d'inférence qui propage les valeurs attendues aux formules. Les utilisateurs peuvent explorer de façon interactive, appliquer, affiner, ou rejeter ces suggestions de changement. Outre les outils d'aide au débogage par les utilisateurs non-informaticien, il existe d'autres approches qui tirent parti des facteurs humains et sociaux. Par exemple, une étude de débogage par utilisateur final montre que lorsque les utilisateurs finaux travaillent en binôme plutôt que seul, ils sont plus systématiques et objectifs dans leurs tests d'hypothèses (Chintakovid et al. 2006). Cette technique est mise en œuvre à travers une approche qui consiste à enregistrer en vidéo les utilisateurs lors d'une tâche de débogage et à utiliser ces enregistrements vidéos pour enseigner des stratégies de débogages, dans lesquels l'utilisateur peut demander une assistance humaine filmée pour débugger son programme (Subrahmaniyan et al. 2007).

2.2.4 La réutilisabilité du code

La réutilisabilité consiste, généralement, à modifier le code (ou le modèle) existant pour l'adapter à un nouveau contexte, ou à construire son application par composition des éléments d'une bibliothèque (Ko et al. 2011). Elle permet de gagner du temps, d'éviter l'écriture à partir du début d'un nouveau code et constitue une bonne base pour la maintenabilité du code (Ravichandran et Rothenberger 2003). Promouvoir la réutilisabilité en programmation par l'utilisateur final, est une activité difficile car les développeurs non-informaticiens ont rarement l'occasion ou la formation nécessaire pour concevoir des programmes hautement réutilisables (Burnett et Scaffidi 2011). De plus, d'un utilisateur à un autre, les priorités et les objectifs du programme à créer peuvent varier. Il est très difficile pour les utilisateurs finaux de construire un programme avec un certain niveau d'abstraction pour faciliter la

réutilisabilité. Pour aider à réduire cette difficulté, des chercheurs commencent à proposer des modèles qui rendent les programmes de l'utilisateur final réutilisables, et pour aider les utilisateurs à rechercher des référentiels pour les programmes réutilisables liés à des intérêts particuliers de l'utilisateur (Scaffidi 2010). En dehors des référentiels, d'autres travaux ont commencé à explorer la façon d'aider les utilisateurs à extraire des éléments réutilisables (Oney et Myers 2009).

2.3 Synthèse sur l'EUD

L'EUD regroupe tous les principes d'EBP (Example Based Programming) qui est une extension du concept des enregistreurs de macros. Ce concept permet à l'utilisateur d'enregistrer et de rejouer ses actions. Cependant, le rejeu des macros se réduit uniquement aux actions enregistrées tandis que dans le cas de l'EBP, le système produit une généralisation des actions enregistrées. La généralisation s'appuie sur les exemples enregistrés pour générer un programme. Les systèmes intégrant les techniques de l'EBP enregistrent, généralisent et rejouent les actions de l'utilisateur à travers des interfaces particulières qui gardent l'objectif fonctionnel de l'application. Ainsi un système d'EBP permet à des utilisateurs non-informaticiens de programmer et répond aux évolutions rapides et à la diversité des besoins toujours croissants des utilisateurs finaux (Burnett et Scaffidi 2011). Les techniques d'EBP permettent aux non-programmeurs d'automatiser les tâches répétitives (Girard et al. 1997), d'adapter une application à leurs besoins spécifiques (Coutaz et al. 2012) et d'intégrer des applications pour développer une solution sur mesure (Piernot et Yvon 1995). Cependant, les utilisateurs de système d'EBP peuvent créer des programmes erronés (Ko et al. 2011; Cao et al. 2010). La question est de savoir comment les utilisateurs peuvent modifier le programme s'ils changent d'avis sur son fonctionnement ou s'ils se trompent. En effet, permettre aux non-informaticiens de créer des programmes de qualité est devenu très important depuis les années 1990, car d'une part, les techniques d'EUD sont de plus en plus répandues, et d'autre part, les programmes créés par les non-informaticiens contiennent souvent des erreurs.

La correction de ces erreurs en utilisant les systèmes d'EBP est impossible car les utilisateurs de systèmes d'EBP sont généralement considérés comme non-programmeurs, il semble inapproprié de les obliger à lire et à comprendre le code obtenu. La difficulté est de comprendre le code généré et de le modifier quand il n'est pas tout à fait correct. S'ils pouvaient comprendre ce code, ils pourraient être en mesure de l'écrire directement, donc l'EBP ne serait peut-être pas nécessaire. Les utilisateurs finaux ne connaissent ni les mécanismes de contrôle de qualité, ni les processus formels de développement, ni les diagrammes de modélisation, ni les critères d'adéquation des tests, et ne peuvent pas prendre le temps d'apprendre de telles choses. L'un des défis importants que doivent relever les systèmes d'EBP est de permettre à l'utilisateur de comprendre et de modifier le programme sans annuler les avantages qu'offre l'EBP (Lieberman 2001). De plus, une attention portée à la qualité est importante pour les systèmes d'EBP car un logiciel (ou un programme) mal écrit peut entrainer une perte de données, des pertes financières, des logiciels non sécurisés etc…, et ce, même si le logiciel est créé par des utilisateurs finaux (Burnett et Scaffidi 2011).

L'intégration des techniques de base de l'EBP dans un cycle de vie (Figure 50), vient combler ces lacunes de l'EBP en proposant d'intégrer la description des exigences, la conception, les tests, la vérification et le débogage dans les techniques existantes permettant aux non-programmeurs de créer des programmes. La conception est largement exploitée par les outils d'EBP et est souvent combinée avec la description des exigences. Plusieurs outils d'EBP

fournissent une intégration étroite entre les tests et le débogage (Burnett et Scaffidi 2011). La plupart des systèmes d'EBP intègrent déjà une phase de vérification qui correspond au rejeu. Ces systèmes ne sont pas forcément conçus dans le respect du cycle de vie d'EUD. Cependant, plusieurs outils liés aux paradigmes de spreadsheet intègrent des techniques de tests pour la détection des erreurs. L'outil le plus avancé intégrant ces techniques est WYSIWYT (What You See Is What You Test) (G. Rothermel et al. 1998). Cet outil est utilisé dans la manipulation des tableurs et permet aux utilisateurs de valider, à n'importe quelle phase du processus de conception, les valeurs qu'ils croient correctes. Les tests sont réalisés dans WYSIWYT de façon incrémentale et systématique. Certaines des étapes du cycle de vie du EUD telles que la réutilisabilité et la maintenabilité ne deviennent évidents que lorsque le programme créé fonctionne pendant un certain temps (Burnett et Scaffidi 2011). D'autres systèmes offrent, en plus des techniques de tests et de vérification, des techniques de débogage qui offrent la possibilité à l'utilisateur de modifier le programme qu'il a créé.

L'ajout de nouvelles fonctionnalités aux logiciels créés par utilisateur finaux et la prise en compte de la qualité ont conduit des chercheurs à proposer un cycle de vie de conception qui est proches de celui classique du Génie logiciel. L'objectif étant d'aider les non-programmeurs à construire des logiciels avec de solides garanties de qualité, sans interférer avec la nature itérative légère du cycle de vie d'EUD. Pour atteindre cet objectif, les non-programmeurs doivent pouvoir tester et déboguer facilement les programmes générés. Il faut donc leur offrir à travers les systèmes d'EUD, des styles d'interactions qui sont les plus naturels possibles. De fait, la plupart des systèmes d'EUD exploitent d'autres styles d'interaction notamment les langages visuels et/ou textuels, dans une certaine mesure. Le langage visuel permet de représenter la sémantique avec de nombreux attributs d'une représentation visuelle tels que la position, la couleur, la taille et l'intersection avec d'autres formes. La programmation avec du texte est, quant à elle, la technique d'interaction la plus traditionnelle pour la programmation, mais pas forcément la plus facile à appréhender pour un non-programmeur.

D'autre part, la qualité des logiciels conçus par les non-programmeurs dépend également du respect des exigences. Partant du principe que les utilisateurs finaux ont une bonne expertise de leur domaine car ils y travaillent tous les jours, on peut considérer qu'ils ont une bonne idée des exigences des systèmes qu'ils utilisent. Ils ne fournissent donc pas d'efforts supplémentaires pour les exprimer, les documenter ou les vérifier (Burnett et Scaffidi 2011). Notre hypothèse est qu'offrir un système d'EUD aux utilisateurs experts (Expert métier qui est dans notre cas l'utilisateur final du système d'EUD) peut nous permettre de récupérer une bonne spécification fonctionnelle du système à concevoir.

3 Proposition d'une démarche de spécification fonctionnelle et de génération d'application de contrôle-commande

De la section précédente, nous déduisons que l'EUD dont les techniques sont décrites dans la section 2.2, s'avère être une solution séduisante permettant à un expert métier de programmer les spécifications fonctionnelles à travers une interface. Cependant, elles doivent être appliquées en respectant le cycle de vie de la Figure 50. En effet, l'utilisation des techniques de l'EUD pour créer un programme assurer la qualité des codes créés. En s'appuyant sur les techniques de l'EUD, nous proposons dans cette section une démarche permettant de faciliter les spécifications fonctionnelles des systèmes complexes.

Il est important de noter que l'appellation EUD a été formalisée car les systèmes d'exemples permettent aux utilisateurs non-programmeur de réaliser des actions sur des objets concrets pour fabriquer des programmes. Aussi le thème « exemple » (en anglais *demonstration*) est utilisé parce que l'utilisateur démontre le résultat souhaité à l'aide des valeurs des exemples. Dans ce mémoire nous utiliserons la formalisation **EBP** (Figure 49) car notre proposition ne s'adresse pas aux « utilisateurs finaux » mais plutôt aux utilisateurs experts (expert métier qui conçoit le schéma PI&D du système de contrôle-commande).

Notre démarche vise à utiliser un système d'EBP composé d'une IHM d'EGRC (Enregistrement Généralisation Rejeu Génération) et de deux modules : un module de généralisation et un module de génération de commandes de haut-niveau. L'*IHM d'EGRC* fournit des interfaces particulières, familières à l'expert métier, pouvant naturellement intégrer les techniques d'EBP tout en gardant l'objectif fonctionnel de l'application et exploite les informations issues des modèles de tâches précédemment obtenus pour guider l'utilisateur expert durant tout le processus de spécification. En effet, dans les modèles de tâches, la sous-tâches « *Manipulation de commandes prévues* » (section 5.2 Chapitre 2) renferme les informations nécessaires permettant de guider le concepteur pour la réalisation des fonctions de haut-niveau.

L'IHM ainsi visible par l'utilisateur expert prend en compte ses commandes de bas-niveau (consignes exprimées sur un élément du procédé représenté sur l'IHM) et lui fait un retour d'information sur l'état du composant sur lequel il vient d'interagir. Notre système d'EBP permet à l'utilisateur expert (non-informaticien) de décrire lui-même les services fonctionnels de haut-niveau de son système, à partir des services élémentaires. Ces services fonctionnels contiennent des interactions élémentaires et des données de configuration. Ainsi l'expert, acteur de la programmation évite beaucoup d'erreurs liées à l'interprétation des spécifications fonctionnelles. L'obtention des spécifications fonctionnelles des systèmes sociotechniques complexes devient ainsi plus facile. Notre objectif est de capturer la connaissance de l'expert (non-informaticien) sur le système à concevoir afin d'avoir les spécifications fonctionnelles.

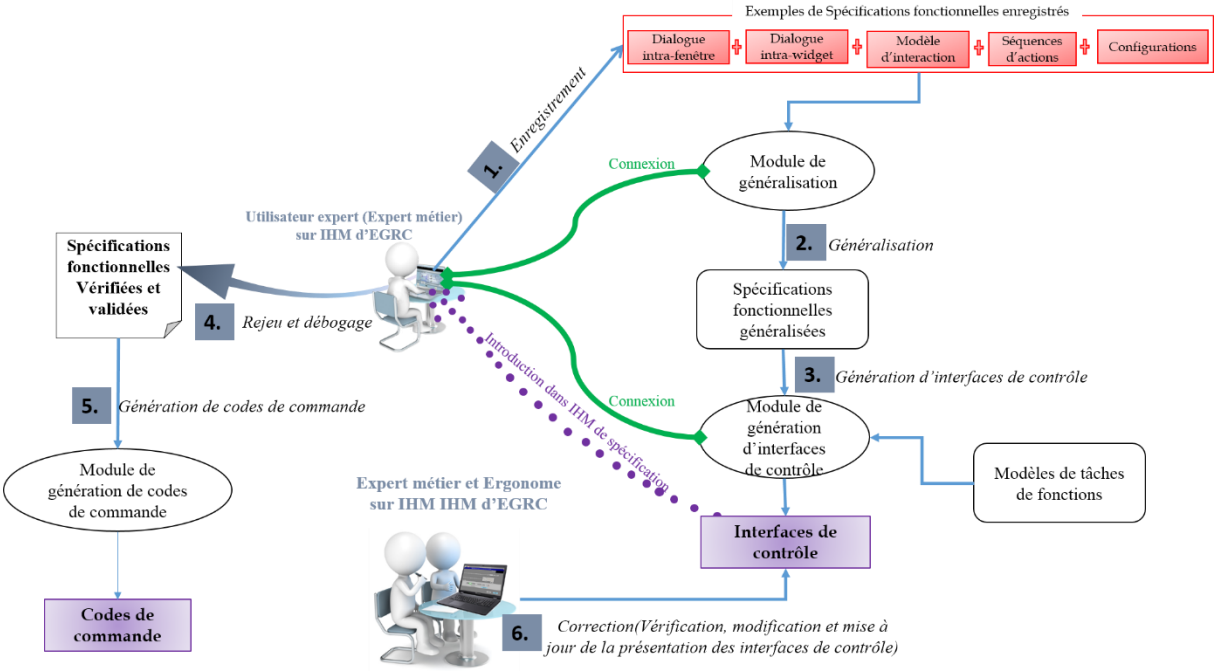


Figure 52 Démarche de spécification fonctionnelle

La démarche de spécification se déroule en six phases présentées sur la Figure 52. Une *première phase d'enregistrement* permet à l'utilisateur expert de spécifier le comportement attendu par le système, de manière interactive, sur une maquette de l'interface de supervision du système.

À partir de quelques exemples ainsi spécifiés, une *deuxième phase de généralisation automatique* peut alors être mise en œuvre. La généralisation permet d'avoir d'une part, une liste exhaustive des configurations (Définition 3) possibles permettant d'exécuter les fonctions spécifiées, et d'autre part, les programmes généralisés.

> Une configuration est associée à une fonction et définit l'état des éléments nécessaire à sa réalisation.

Définition 3 Configuration

Les programmes génériques et toutes les configurations possibles généralisées constituent les spécifications fonctionnelles du système à concevoir. Ces spécifications doivent être vérifiées et validées. Pour ce faire, nous proposons un rejeu (test, vérification et la validation) des spécifications fonctionnelles sous forme de *visualisation* (voir section 2.2.2). Cette forme de vérification s'appuie sur les expertises du domaine ou la connaissance que possèdent les experts du fonctionnement du système. Elle permet aux experts non-informaticiens de visualiser le comportement du programme créé afin de vérifier plus facilement sa conformité. Pour ce type de vérification, les spécifications fonctionnelles doivent être présentées au concepteur à travers une interface qui lui est familière. Nous proposons, à cet effet, une démarche permettant dans premier temps, d'exploiter les données issues des spécifications fonctionnelles généralisées pour générer les commandes de haut-niveau (Définition 4), démarche décrite dans la *troisième section de ce chapitre*, puis d'utiliser ces commandes de haut-niveau pour vérifier et/ou corriger, puis valider les spécifications fonctionnelles.

> Nous définissons une commande de haut-niveau (commande de lancement d'une fonction) comme un ensemble de commandes élémentaires (ouverture ou fermeture d'un élément) exécutées dans un ordre précis pour répondre au besoin exprimé par une fonction. Une commande de haut-niveau est composée d'une interface de contrôle et de codes de commandes. Elle permet à un opérateur de lancer une fonction du système, à partir de la supervision, en s'appuyant sur une configuration du système et sur une séquence d'exécution précise.

Définition 4 Commande de haut-niveau

Une *quatrième phase de rejeu et débogage* permet donc à l'utilisateur expert de vérifier, valider et corriger (s'il y a lieu) les spécifications fonctionnelles généralisées à travers les commandes de haut-niveau précédemment générées. Les spécifications fonctionnelles vérifiées et validées sont ensuite utilisées pour mettre à jour les commandes de haut-niveau.

Une dernière phase de vérification, modification et mise à jour des interfaces de contrôle est réalisée par l'utilisateur expert et l'ergonome afin d'assurer l'utilisabilité des interfaces de contrôle permettant de lancer les commandes de haut-niveau, générées par le système d'EBP.

Nous détaillons ces différentes phases dans les sections suivantes.

3.1 La phase d'enregistrement

La phase d'enregistrement permet de recueillir des exemples de séquences d'exécution pour toutes les fonctions (exemples de spécifications fonctionnelles). Lors de l'enregistrement de ces séquences, les informations relatives aux objets d'interactions ainsi que les dialogues fonctionnels liés à ces objets proposés pour la réalisation des tâches issues du modèle de tâches sont enregistrés en tant que *modèles d'interaction*, de dialogues correspondants à l'action de l'utilisateur (*Dialogue intra-widget* voir section 1.2.3) et à la réaction du système (*Dialogue intra-fenêtre* voir section 1.2.1). Les exemples de spécifications fonctionnelles contiennent également les *programmes enregistrés (séquences d'actions)* correspondant aux actions de l'utilisateur sur l'interface et ainsi que les *configurations* représentant l'état des éléments du système manipulées lors de l'enregistrement des exemples.

3.2 La phase de généralisation

La généralisation reste un problème central et complexe de l'EBP. Dans les systèmes d'EBP, on distingue deux types de généralisation : la généralisation utilisant les techniques d'inférence et la généralisation sans inférence. Les systèmes utilisant ces techniques peuvent produire des actions incorrectes même lorsque l'utilisateur ne fait aucune erreur lors de la description des exemples (B. a Myers et Mcdaniel 2000). Bien qu'il existe différentes techniques de généralisation, elles ne sont pas adaptées à la généralisation complète dans le cadre de la conception des systèmes de supervision industrielle complexes. Par exemple, concevoir une base de connaissances pour utiliser des méthodes d'inférence qui permettraient de généraliser les configurations, peut être une tâche fastidieuse. La complexité des systèmes traités nous a conduits à proposer deux types de généralisations réalisées au cours de la deuxième phase : la généralisation des programmes et la généralisation des configurations.

3.2.1 La généralisation du programme

La généralisation du programme consiste à comparer les programmes enregistrés lors de la description des exemples, pour remplacer les variables par des paramètres (ou des valeurs par des variables). Cette première généralisation produit des programmes génériques pouvant être utilisés dans différents contextes.

3.2.2 La généralisation des configurations

La généralisation des configurations quant à elle, se base sur la théorie des graphes. Tout d'abord l'utilisation d'un algorithme de combinaison permet de proposer à l'utilisateur tous les chemins possibles qu'un système peut utiliser pour atteindre l'objectif fonctionnel demandé. Ces chemins possibles sont validés par l'expert métier puis combinés avec l'utilisation d'un solveur à un algorithme de configuration. Le but est de fournir l'exhaustivité des possibilités de configurations permettant l'exécution des fonctions spécifiées. Pour cela, l'utilisation du solveur issu des travaux précédents (Bignon 2012) est combinée à deux algorithmes implémentés : un algorithme de combinaison et un algorithme de configuration.

3.3 Génération d'interfaces de contrôle : Proposition de génération conjointe de la présentation et des trois types de dialogue

Pendant le pilotage des systèmes de contrôle-commande, les fonctions sont lancées depuis l'IHM de supervision. Elles disposent donc d'une représentation sur l'IHM. Nous appelons cette représentation *interface de contrôle pour commande de haut-niveau*. Une *interface de contrôle*

pour commande de haut-niveau peut être qualifiée d'interface utilisateur car elle permet à l'opérateur d'interagir avec le système.

Générer les commandes de haut-niveau revient à générer conjointement l'interface de contrôle et les codes de commande correspondant à cette interface. L'interface de contrôle est composée d'un ensemble de widgets pour la supervision, représentant les fonctions sur l'IHM complète, et de codes de commandes de haut-niveau décrits avec des grafcets qui communiquent avec le noyau fonctionnel de l'IHM de supervision.

La conception des commandes de haut-niveau est identique à la conception d'un système interactif. Au workshop de Seeheim (Pfaff 1983), l'architecture d'un système interactif est composée de trois parties (Figure 53) : la présentation, le dialogue et le noyau fonctionnel.

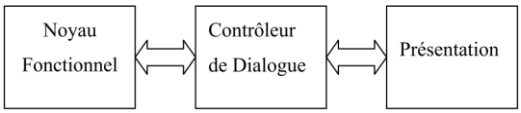


Figure 53 Le modèle de Seeheim (Samaan 2006)

Dans cette section nous proposons une démarche permettant de générer la présentation, les trois types de dialogue et le code de commande de haut-niveau des systèmes de contrôle-commande.

3.3.1 Génération des modèles de dialogue

3.3.1.1 Le dialogue inter-fenêtre

Pour déduire le dialogue inter-fenêtre, nous nous sommes inspirés des algorithmes utilisés dans les simulateurs de tâches et de certaines des heuristiques proposées l'outil CTTE (Samaan 2006). Nous analysons l'arbre de tâches à partir de la tâche principale pour rassembler les tâches systèmes et interactives élémentaires en fonction de leurs décompositions (alternative, parallèle, séquentielle, élémentaire etc.). Dans un premier temps nous utilisons les mêmes règles que dans les travaux précédents en précisant que deux tâches séquentielles ne doivent pas appartenir au même PTS (ensemble de présentations).

Nous appliquons ensuite les heuristiques *Sharing most elements Sets* et *Singe Element Sets*, qui indiquent respectivement que certains PTS qui partagent le plus d'éléments sont unifiés et que si un PTS est composé de seulement 1 élément, il est associé à un autre PTS.

Après l'application de ces deux heuristiques, nous obtenons **les ensembles de présentation** correspondant à une commande de haut-niveau. Ces deux ensembles sont reliés par des transitions exprimées avec des règles en se basant sur la décomposition séquentielle et l'ordre des tâches. Ces transitions représentent **les dialogues inter-fenêtres**.

Les modèles de tâches ne contiennent pas tous les éléments nécessaires à l'expression complète du dialogue des applications interactives actuelles. Bien que la simulation ressemble au dialogue des applications, certaines considérations ergonomiques imposent l'enrichissement du modèle de dialogue inter-fenêtre des applications obtenues à partir du modèle de tâches (Palanque, Bastide, et Winckler 2003). Cet enrichissement peut être décrit dans les *dialogues intra-fenêtre et intra-widget*.

3.3.1.2 *Le dialogue intra- widget*

Nous avons enregistré puis généralisé le dialogue intra-widget lorsque l'utilisateur spécifie des exemples de fonctions. Il s'agit d'obtenir sous forme de programme les actions de l'utilisateur sur le système. Ce dialogue contient d'une part les données enregistrées, et d'autre part, les scripts (programmes) généralisés.

La déduction de ce dialogue par le système d'EBP qui exploite les données des modèles de tâches permet de maintenir une **cohérence** dans le processus de génération et de conserver le lien entre le modèle de tâches et de dialogue intra-widget. En effet, les tâches dans les modèles de tâches sont utilisées pour guider l'utilisateur pendant l'enregistrement des exemples lors de l'opération spécification fonctionnelle (Voir section 3.1).

3.3.1.3 *Le dialogue intra-fenêtre*

Comme pour le dialogue intra-widget, nous proposons d'utiliser le système d'EBP contenant les informations sur les modèles de tâches, pour obtenir le dialogue intra-fenêtre. En effet, la spécification du dialogue des widget d'une même fenêtre est réalisée pendant l'enregistrement des exemples sur l'IHM de spécification. Ces dialogues sont enregistrés sous forme de scripts et décrivent la réponse du système lorsque l'utilisateur finit la réalisation de la tâche qui lui est indiquée par l'interface.

L'utilisation de ce dialogue permet entre autre d'afficher les widgets de façon plus ergonomique.

3.3.1.4 *Synthèse sur la génération du dialogue*

Alors que la plupart des systèmes de génération basés sur les modèles de tâches ne s'adressent qu'au dialogue inter-fenêtres, notre proposition permet d'assurer la génération du dialogue à tous les niveaux. D'une part, nous générons le dialogue inter-fenêtre à partir des modèles de tâches. Et d'autre part, avec l'utilisation des techniques d'EBP qui elles-mêmes exploitent les modèles de tâches, nous obtenons les dialogues intra-fenêtre et intra-widget.

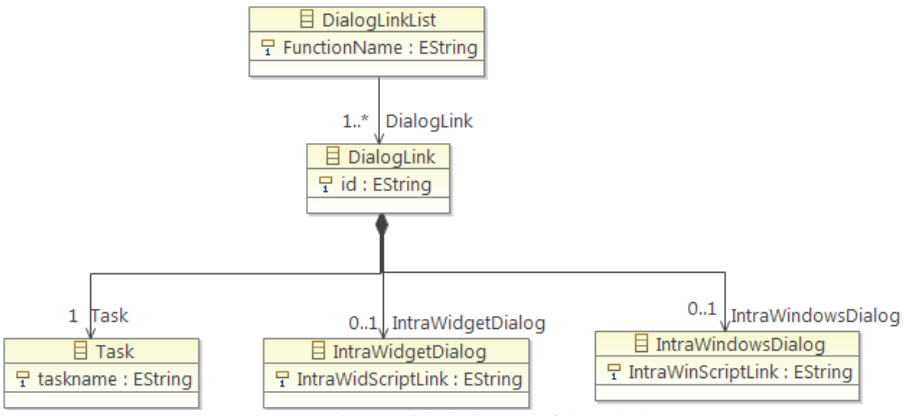


Figure 54 Méta-modèle de liaison tâches-dialogue

Pour assurer les liens entre les tâches issues des modèles de tâches et les dialogues intra-fenêtre et intra-widget, en plus des scripts correspondants à ces deux dialogues, nous enregistrons un modèle de liaison tâches-dialogues qui est conforme au méta-modèle de la Figure 54. Sur cette Figure 54 un ensemble de dialogues (*DialogLinkList*) est rattaché à une fonction par l'attribut « *FunctionName* » qui est le nom de la fonction dont on veut générer l'interface de contrôle. La classe *DialogLinkList* met en relation les attributs *IntraWidgetScriptLink* et

IntraWindowsScriptLink qui sont des URI des différents dialogues enregistrés avec le nom de la tâche correspondante.

3.3.2 Génération du modèle de présentation

La conception de la présentation des interfaces interactives est souvent complexe et nécessite une bonne expertise. Il est donc important d'identifier les modèles déclaratifs et les mécanismes d'inférence pour réduire considérablement l'effort de conception (Paterno, Breedvelt-schouten M., et Koning 1999). L'obtention des ensembles de présentations (PTS voir section 3.3.1.1) est la première étape de génération du modèle de présentation. Cependant, il n'y a aucune indication relative aux widgets permettant à l'utilisateur de réaliser les tâches contenues dans ces PTS. Pourtant, pour construire une interface complète, il est nécessaire de connaître quelle interaction est utilisée. Il faut faire des choix à propos de l'interaction. Il est donc nécessaire de prévoir un modèle intermédiaire qui permet la liaison des interacteurs aux actions du modèle de tâches contenues dans les PTS. Nous nommons ce modèle intermédiaire *modèle d'interaction*.

3.3.2.1 Le modèle d'interaction et son méta-modèle

Pour utiliser un système interactif, l'utilisateur doit avoir à sa disposition un ou plusieurs dispositifs d'interaction, les *widgets*. Ces *widgets* l'aident à interagir avec le système en employant différentes techniques d'interaction. Une technique d'interaction décrit dans une IHM la manière de se servir d'un dispositif d'interaction pour accomplir une tâche générique (Hinckley et al. 2014). Les techniques d'interaction ne sont pas forcément liées à un dispositif spécifique, ni à une tâche spécifique. Cependant, certaines techniques d'interaction restent mieux adaptées à certains dispositifs qu'à d'autres et à certaines tâches qu'à d'autres (Samaan 2006). Le *modèle d'interaction* nous permet d'obtenir les widgets les mieux adaptés pour réaliser les tâches interactives élémentaires décrites dans les modèles de tâches.

Dans ses travaux, (Samaan 2006) a proposé un modèle d'interaction pour faire le lien entre les actions de l'utilisateur et le modèle du domaine. Pour la description et la manipulation de ce modèle, il s'est basé sur l'architecture AMF (Agent Multi-Facette) qui est une architecture multi-agents multi-facettes. Le rôle de son modèle d'interaction est d'assurer la gestion de la communication entre l'utilisateur et le système et entre les composants du système lui-même.

Le *modèle d'interaction* que nous proposons ici permet de faire le lien entre les modèles de tâches et les widgets possibles de l'interface de contrôle. Ce modèle fera la liaison entre les objets d'interactions et les tâches du modèle de tâches. Nous proposons de déduire ce modèle des phases de spécification des exemples de chaque fonction (Voir 3.1).

Les différentes étapes de spécification qui peuvent nous permettre de déduire le type de widget qui sera utilisé, sont celles effectuées lors de la phase de configuration du circuit. Le modèle de tâches est exploité à ce niveau pour guider le spécificateur lors de l'enregistrement de chaque fonction. Plusieurs règles sont introduites dans notre outil de spécification pour déduire le modèle d'interaction. Elles seront décrites au chapitre 5.

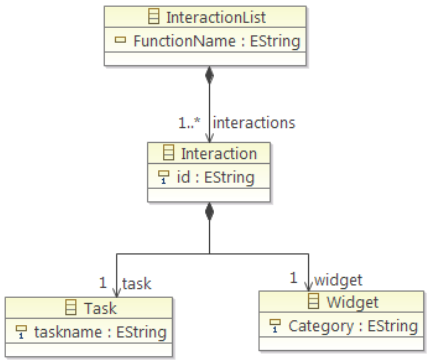


Figure 55 Un métamodèle pour le modèle d'interaction décrit avec EMF

Les informations déduites à partir des différentes phases de spécifications sont regroupées dans le *modèle d'interaction*. Le modèle d'interaction contient tous les objets d'interactions nécessaires aux tâches de réalisation d'une fonction. Ce modèle est conforme au méta-modèle de la Figure 55. Un modèle d'interaction (*InteractionList*) contient une liste des interactions possibles associées à une fonction. Il contient donc une ou plusieurs interactions. Chaque interaction a un attribut id et est composée d'informations *taskname* issues du modèle de tâches et *category* de widgets manipulés référencés dans une bibliothèque de Widgets. Nous avons donc défini une bibliothèque de widgets spécifiques.

3.3.2.2 Une bibliothèque de widgets spécifiques

Dans la littérature, il existe plusieurs boites à outils de widgets telles que *GUI toolkit*, *layout manager*, *Swing (java)* qui sont utilisées pour la description de la présentation des applications (Ramón et al. 2015). Cependant, ces boites à outils ne sont pas adaptées à la conception des systèmes de contrôle-commande car ce type de système est construit avec des logiciels spécifiques type SCADA (Supervisory Control And Data Acquisition).

Généralement, les widgets permettent d'effectuer des tâches élémentaires d'interactions. Pour aider l'opérateur à réaliser les tâches de supervision, nous avons défini une bibliothèque de widgets conçus avec un logiciel type SCADA. La définition des widgets avec ce logiciel est justifiée par le fait que le système de supervision est conçu avec ce logiciel. L'intérêt de la définition des widgets est qu'ils permettent d'établir des styles d'interaction standard pour faciliter l'apprentissage et la réutilisation de code.

Pour formaliser et définir une bibliothèque de widgets, nous nous sommes assurés de la définition des principaux types de widgets en nous basant sur les formalisations faites dans la bibliothèque Qt (Qt 2016). Qt est une bibliothèque dédiée au développement des applications multiplateformes et plus spécifiquement des interfaces graphiques (GUI). Les principaux types de widgets définis dans la bibliothèque Qt sont : les fenêtres, les boutons, les afficheurs, les champs, les conteneurs. Nous nous sommes appuyés sur ces principaux types pour formaliser notre bibliothèque de widgets.

Les widgets de type *fenêtre* sont des widgets qui ne sont contenus dans aucun autre widget. Par exemple, un widget qui n'a pas de parent sera considéré comme fenêtre. Une boîte de dialogue est une fenêtre, généralement de petite taille, dans laquelle il y a peu d'informations.

Les widgets de type *Button (bouton)* se distinguent en trois catégories : les *PushButton* (les boutons classiques), les *CheckButton* (les boutons case à cocher, on considère que c'est un bouton en conception GUI) et les *RadioButton* (les boutons radio). Nous connaissons tous les

boutons classiques car c'est l'élément le plus classique et le plus commun des fenêtres. Un bouton case à cocher est quant à lui, généralement associé à un texte de libellé. Un bouton radio permet de faire un choix parmi une liste. C'est une case à cocher particulière car une seule case peut être cochée à la fois parmi une liste. Les boutons radio qui ont le même widget parent sont mutuellement exclusifs. Si un est coché, les autres seront automatiquement décochés.

Les widgets de type *afficheurs (display)* permettent d'afficher du texte ou une image ou encore une barre de progression.

Les widgets de type *Champs (Field)* permettent de saisir des données. Ce type de widgets est organisé en plusieurs catégories : *Edit* (un champ de saisie d'un texte, d'un nombre entier ou réel), *Slider* (un curseur qui permet de sélectionner une valeur numérique) et *drop-down list* (liste déroulante), *TextBox* etc.

Les widgets de type *Conteneur (containers)* ont été créés spécialement pour pouvoir en contenir d'autres.

Le Tableau 4 présente un récapitulatif des types et catégories de widgets définis pour notre bibliothèque de widgets. La bibliothèque de widgets regroupe l'ensemble des widgets nécessaires à la conception d'un système interactif de contrôle-commande.

Type de widget	Windows	Button	Display	Field	container
Catégorie de widget		PushButton	TextDisplay	Edit	
		CheckButton	ImageDisplay	Slider	
		RadioButton	ProgressBar	drop-down list	
				TextBox	

Tableau 4 Récapitulatif des types et catégories de widgets

Afin de pouvoir utiliser la bibliothèque dans notre processus de génération des interfaces de contrôle, basé sur les techniques de l'IDM, nous avons défini un méta-modèle pour la bibliothèque de widgets (Figure 56), afin de permettre l'expression des règles de transformations de modèles.

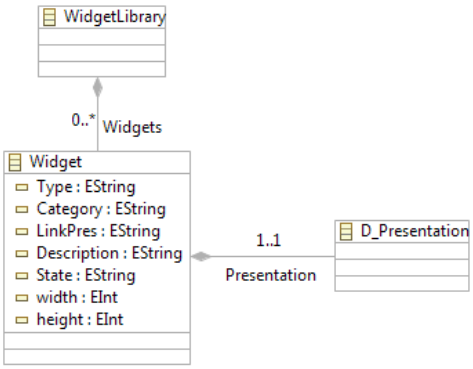


Figure 56 Métamodèle de la bibliothèque de widgets décrit avec EMF

Sur cette Figure 56, la bibliothèque de widgets (WidgetLibrary) contient des widgets. Chaque widget est caractérisé par les attributs *Type*, *Category*, *Description, State, width, height* et est composé d'une présentation rattachée au widget par l'attribut *LinkPres* qui est un URI[17].

La Figure 57 présente la structure de la bibliothèque de widgets. Toutes les données sont définies dans un fichier qui est conforme au métamodèle défini sur la Figure 56. Chaque widget de la bibliothèque dispose d'un sous-répertoire contenant un répertoire pour *Presentation* et un répertoire pour *Dialog*. Le répertoire *Dialog* est composé de trois sous-répertoires *Intra-widget dialog*, *Intra-window dialog* et *Inter-window dialog*. Le répertoire *Presentation* contient les fichiers décrivant la vue du widget. Le répertoire *Dialog* contient une partie des données dérivées des spécifications (de tâches interactives de l'utilisateur sur le système et la réaction du système par rapport à ces tâches), qui sont liées au widget.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lib:Library xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:lib="http
    <Widget>
        <Type>Button</Type>
        <Category>PushButton</Category>
        <Description>bouton classique qui permet de faire un clic</Description>
        <width>136</width>
        <height>40</height>
        <LinkPres>PushButton\Presentation\</LinkPres>
    </Widget>
    <Widget>
    <Widget>
    <Widget>
        <Type>Field</Type>
        <Category>drop-down list</Category>
        <Description>Faire un choix dans une liste</Description>
        <width>88</width>
        <height>27</height>
        <LinkPres>drop-down list\Presentation\</LinkPres>
    </Widget>
    <Widget>
</lib:Library>
```

Figure 57 Structure de la bibliothèque de widgets

La seule connaissance des widgets utilisables sur l'interface et des PTS ne suffit pas à la conception de la présentation car la présentation qui permet de structurer les différents widgets d'une interface contrôle n'est pas encore définie. La structure des widgets est essentielle pour la génération des interfaces de contrôle. Cette structure peut varier en fonction de l'application. Pour ce faire, nous proposons un modèle pour la structure de la présentation que nous appelons *modèle d'interface de fonction*. Ce modèle sera générée automatiquement à partir des modèles précédemment obtenus (PTS, modèle d'interaction, bibliothèque de widgets). Pour permettre cette génération nous avons défini le méta-modèle auquel il est conforme.

3.3.2.3 Un méta-modèle pour le modèle d'interface de fonction

Le modèle d'interface de fonction permet d'avoir une idée de la structure visuelle des interfaces de contrôle. Il permet de décrire la perception humaine de l'agencement des widgets sur une interface et est un élément clé permettant au contenu d'une vue d'être adapté à différents affichages. La Figure 58 est un modèle de structure d'interface simplifié présenté dans les travaux de (Ramón et al. 2015). Ce modèle est utilisé pour représenter la structure logique d'une vue (c'est-à-dire la hiérarchie des éléments du GUI). Sur la Figure 58, tous les éléments sont considérés comme des widgets ; les *SingleWidgets* tels que *TextBoxes* et *Buttons* sont des éléments non composés. Les containers tels que *Views* (exemple fenêtre ou page web) ou *Panels* sont des éléments qui sont utilisés pour grouper visuellement les autres éléments.

[17] Uniform Ressource Identifier (dans notre cas il s'agit d'une chaine de caractère représentant le chemin d'accès à la présentation et aux dialogues).

Tous les éléments d'une vue qui sont localisés par les coordonnées *X* et *Y*, ont une largeur et une hauteur.

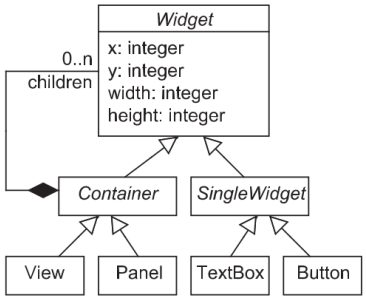


Figure 58 Modèle basique de GUI (Roman et al, 2015)

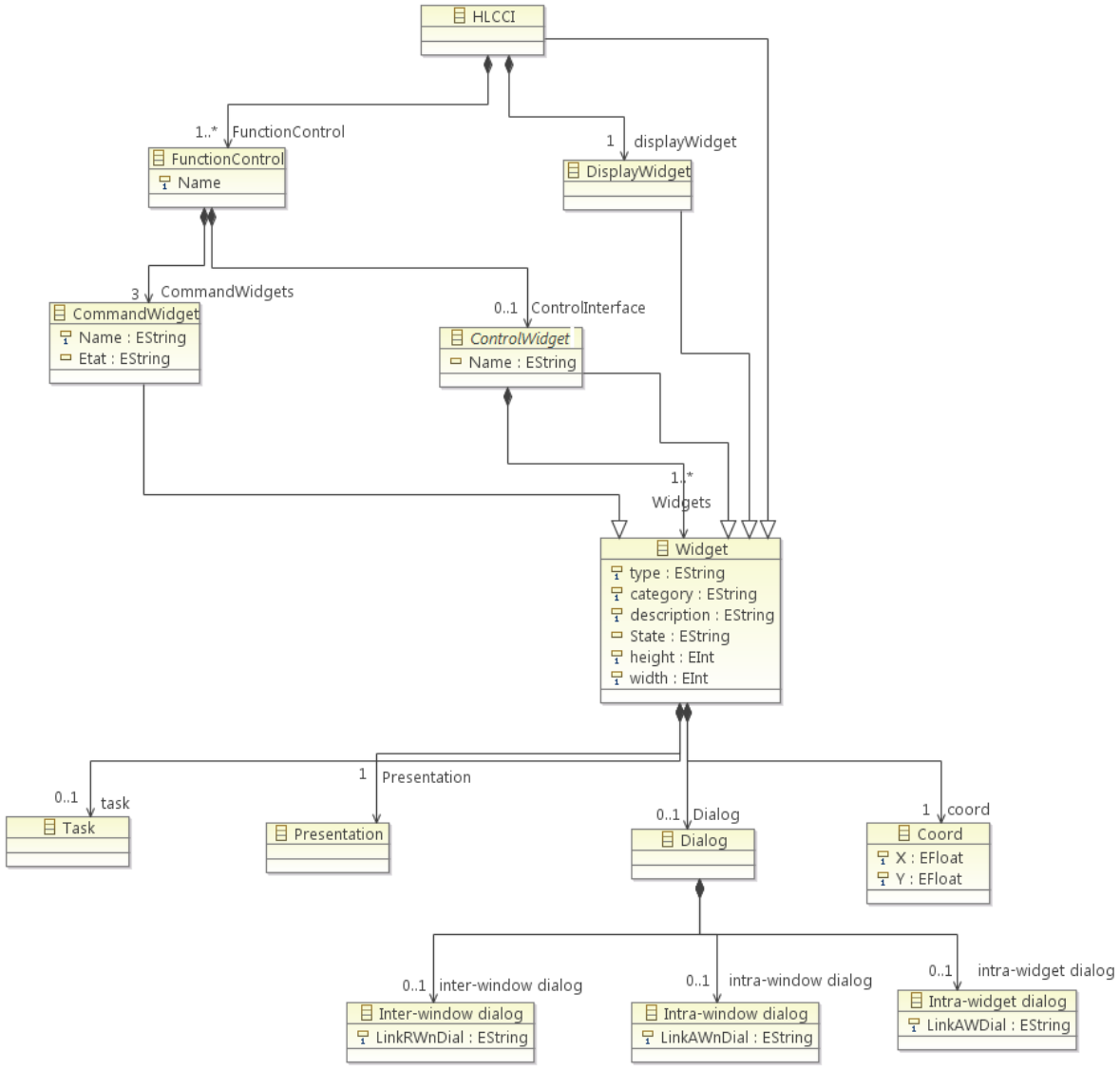


Figure 59 Un métamodèle pour le modèle d'interface décrit avec EMF

En s'inspirant du modèle d'interface basique de la Figure 58, nous pouvons définir un méta-modèle pour notre modèle d'interface de fonction (Figure 59). Sur la Figure 59, HLCCI (High Level Control Command Interface) est un widget de type *conteneur* qui contient toutes les commandes des fonctions de haut-niveau du système et aussi un *DiplayWidget* qui est un widget de type *Diplay (afficheur)* pour afficher des informations de haut-niveau. Une

FunctionControl est liée à une fonction et a un attribut *Name* qui est le nom de la fonction. Une *FunctionControl* est composée de 3 *CommandWidget* et de 0 ou 1 *ControlInterface* (le bandeau de contrôle). Chaque *CommandWidget* est un widget de type *Button* qui permet soit de démarrer la fonction, soit d'arrêter la fonction, soit d'attendre que l'action en cours finisse (une commande grisée). Chaque *ControlInterface* contient 1 ou plusieurs *Widgets*. Un *ControlInterface* est aussi un *Widget* de type *Container* qui contient d'autres widgets permettant de lancer une fonction. Les Widgets sont disposés sur l'interface par rapport aux attributs *X* et *Y* qui représentent leurs coordonnées. Chaque widget est caractérisé par les attributs *Type*, *Category*, *Description, State, width, height* et est composé d'un sous-répertoire contenant un répertoire pour *Presentation* et un répertoire pour *Dialog*. Le répertoire *Dialog* est composé de trois sous-répertoires *Intra-widget dialog*, *Intra-window dialog* et *Inter-window dialog*. Le répertoire *Presentation* contient les fichiers décrivant la vue du widget. Le répertoire *Dialog* contient une partie des données dérivées des spécifications (de tâches humaines et fonctionnelles), qui sont liées au widget. Ces différents répertoires sont rattachés aux widgets par les attributs *LinkPres*, *LinkAWdDial*, *LinkAWnDial*, *LinkRWnDial* qui sont des URI[18].

Les modèles décrits dans les sections 3.3.2 nous permettent d'avoir une idée des widgets possibles utilisables sur une IHM de supervision et de savoir comment est structurée une interface de contrôle à partir de son méta-modèle. Ainsi, nous pouvons en utilisant les techniques de l'IDM, des règles de transformations de modèles, générer un modèle d'interface de contrôle pour la présentation. Nous allons définir des règles par rapport à la structure du GCCI. Ces règles vont nous permettre d'exprimer que :

- Les *commandWidgets* regroupent des widgets de type *Button* permettant de Démarrer, d'arrêter une fonction et un bouton Démarrer en mode « inactif » auquel aucun dialogue n'est lié ;
- Le *controlInterface* est un widget de type container qui contient les widgets décrits à partir du modèle d'interaction.

La structure visuelle est détectée en analysant les positions de tous les éléments dans l'ordre d'affichage pour reconnaître la façon dont ils sont regroupés visuellement. Cette analyse est faite avec le modèle de dialogue intra-fenêtre.

3.3.2.4 Synthèse de la génération des interfaces de contrôle

La démarche mise en œuvre pour l'obtention des interfaces de contrôle se déroule en quatre grandes étapes. La *première étape* concerne la génération du dialogue inter-fenêtre et des ensembles de présentation (PTS) à partir des modèles de tâches telle que détaillée dans la section 3.3.1.1. Les dialogues intra-fenêtre et intra-widget sont issus de l'opération de spécification fonctionnelle.

La *deuxième étape* vise à utiliser le modèle d'interaction, la bibliothèque de widgets et les PTS pour générer le modèle de présentation. À partir du nom des widgets suggérés dans le modèle d'interaction, nous récupérons le widget correspondant à ce nom dans la bibliothèque de widget, pour enrichir les PTS et obtenir le modèle de présentation.

Au cours de la *troisième étape*, les dialogues intra-fenêtre et intra-widget sont liés aux widgets contenus dans le modèle de présentation (voir section 3.3.2.1) pour produire une collection de widgets suggérés. En effet, à partir du nom de la tâche dans le modèle de présentation, nous

[18] Uniform Ressource Identifier (dans notre cas il s'agit d'une chaine de caractère représentant le chemin d'accès à la présentation et aux dialogues).

lions le dialogue (intra-fenêtre ou intra-widget) aux widgets contenus dans le modèle de présentation en nous appuyant sur le nom des tâches. Le modèle de collection de widgets suggérés correspond au méta-modèle de la bibliothèque de widgets. Chaque widget de cette collection dispose d'une présentation et d'un ensemble de dialogues.

La dernière étape vise à générer les interfaces de contrôle. Elle reçoit en entrée la collection de widgets suggérés et les dialogues inter-fenêtre et produit par des règles de transformation de modèle, les interfaces de contrôle conformes au méta-modèle de la Figure 59.

3.4 La phase de rejeu pour la vérification, correction et validation des commandes de haut-niveau générées

Les interfaces de contrôle ont été générées par une combinaison des modèles de tâches avec des informations issues des spécifications fonctionnelles obtenues en utilisant les techniques d'EBP. Lors de la phase de rejeu, l'utilisateur expert manipule les interfaces de contrôle afin de vérifier et valider les configurations et les codes générés.

La phase de rejeu proposée permet de vérifier, détecter et corriger les erreurs dans les configurations généralisées, puis valider pour n'avoir que des configurations vérifiées et validées par l'utilisateur expert (expert métier). Il s'agit d'offrir la possibilité à l'utilisateur expert de vérifier et modifier les configurations généralisées, si elles ne lui conviennent pas, et d'enregistrer les configurations valides.

A l'issue de cette phase, l'utilisateur expert n'enregistre que les spécifications fonctionnelles vérifiées et validées. Ces spécifications sont ensuite utilisées pour générer les codes de commande représentant le noyau fonctionnel connecté aux interfaces de contrôle précédemment générées. L'utilisation à cette étape des spécifications fonctionnelles vérifiées et validées permet d'éviter que des erreurs se propagent dans la conception.

3.5 La génération des codes de commandes de haut-niveau

Pour que le noyau fonctionnel puisse répondre aux actions de l'utilisateur, il existe, pour chaque commande de haut-niveau, un code de commande écrit pour la partie commande qui communique avec le niveau fonctionnel de la supervision. Les codes de commande fonctionnent sur des systèmes d'acquisition de données et de contrôle de l'installation supervisée. Ces systèmes assurent plusieurs fonctions comme la gestion d'entrées/sorties, l'automatisation, la régulation, qui peuvent être dévolues à un logiciel spécifique, généralement un automate.

En raison de la complexité des systèmes de contrôle-commande, l'implémentation manuelle des codes de commande est longue et fastidieuse (plus le nombre de commandes de haut-niveau est important, plus il faut du temps pour les implémenter manuellement).

Dans la littérature, plusieurs approches ont été utilisées pour la génération de codes de commandes(VEPSÄLÄINEN, HÄSTBACKA, et KUIKKA 2008; Ferrarini et al. 2009; Bévan 2013). Malgré leurs nombreux avantages, ces approches ne sont pas adaptées aux lois de commandes utilisées pour la conception des systèmes de supervision industrielle embarqués sur les navires. En effet, le code de commande généré doit être transformé en langage normalisé avant d'être utilisé. Généralement, le grafcet (**GRA**phe **F**onctionnel de **C**ommande **E**tape **T**ransition) est utilisé pour construire le code de commande des systèmes de supervision industrielle. Le grafcet est un outil graphique de description du comportement attendu de la partie commande.

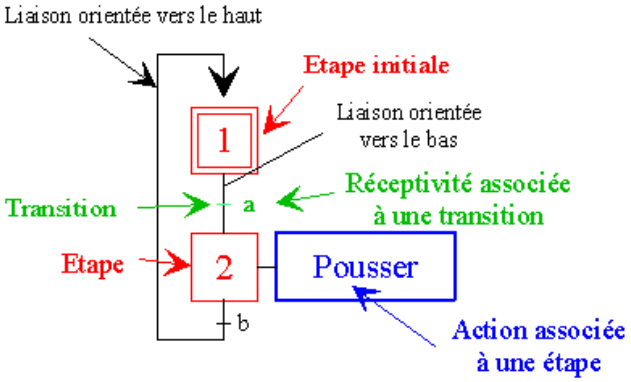


Figure 60 Exemple de représentation du grafcet

Depuis 1988, le grafcet est un outil de description normalisé (Norme C.E.I. 848) qui fonctionne en logique séquentielle. C'est un outil simple qui permet les représentations fonctionnelles, opérationnelles et technologiques de la plupart des automatismes industriels. La description du fonctionnement d'un automatisme logique peut alors être représentée graphiquement (Figure 60) par un ensemble :

- d'***étapes*** auxquelles sont associées des ***actions***,
- de ***transitions*** auxquelles sont associées des ***réceptivités***,
- de ***liaisons*** (ou ***arcs***) ***orientées***.

3.5.1 Le langage de programmation de Grafcet

Le langage de programmation SFC (Sequential Function Chart) de la norme CEI 61131-3 est un langage graphique de programmation des API (Automates Programmable Industriel) défini par l'IEC en 1993. Cette norme spécifie la syntaxe, la sémantique et la représentation des langages de programmation devant être utilisées pour les API. Elle permet de donner aux programmes une structure proche de la spécification Grafcet par l'utilisation du diagramme fonctionnel en séquence. Ce diagramme offre des éléments pour construire graphiquement les actions, les transitions, les liaisons orientées etc. qui constituent le Grafcet.

3.5.2 Les actions

Pour chaque étape du Grafcet, l'action ou les actions associées sont précisées et sont à effectuer lorsque l'étape est active. Une étape est dite active lorsqu'elle correspond à une phase "en fonctionnement", c'est-à-dire qu'elle effectue l'action qui lui est associée. L'action indique, dans un rectangle, comment agir sur la variable de sortie, soit par assignati on (action continue), soit par affectation (action mémorisée).

Les actions composant le Grafcet d'une fonction de haut-niveau dans les systèmes de contrôle-commandes permettent le lancement de consignes d'ouverture/fermeture ou de marche/arrêt des éléments du système. Ces consignes se traduisent par l'attribution d'un état 1 (ouvert, marche) ou d'un état 0 (fermé, arrêt) aux variables d'entrées envoyées par la supervision, provoquant ainsi un changement d'état des variables de sorties associées.

Par ailleurs, toutes les variables intervenant dans l'exécution d'une commande globale doivent être déclarées. Néanmoins, selon la configuration choisie, ces variables n'interviennent pas toujours dans l'exécution de la fonction.

3.5.3 Les transitions

Les transitions permettent, quant à elles, le passage ou non d'une action à une autre selon l'état de certaines variables. Elles sont représentées par une barre perpendiculaire aux liaisons orientées qui la relient aux étapes précédente(s) et suivante(s). Pour franchir une transition, il faut que les étapes immédiatement précédentes soient actives et que la réceptivité associée à la transition soit vraie. Le franchissement d'une transition entraîne simultanément l'activation des étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.

À chaque transition est associée une réceptivité inscrite à droite de la barre de transition. La réceptivité associée à une transition est une fonction logique qui permet de distinguer parmi toutes les combinaisons d'informations disponibles celle qui est susceptible de faire passer le système aux étapes suivantes : des entrées (capteurs, commande opérateur envoyée depuis la supervision), etc.

3.5.4 Liaisons orientées

Les liaisons orientées relient les étapes aux transitions et les transitions aux étapes. Elles sont représentées par des lignes, parcourues par défaut de haut en bas ou de gauche à droite. Dans certains cas, elles sont représentées par des flèches pour préciser l'évolution du Grafcet pour éviter tout risque de confusion. L'utilisation des flèches peut permettre une meilleure compréhension du Grafcet.

Lorsqu'une liaison orientée doit être interrompue, en raison de la complexité de la fonction à représenter par exemple, le repère de l'étape de destination doit être indiqué.

3.5.5 Description de la génération des codes de commande

Générer les codes de commande revient donc à générer les actions, les transitions ainsi que les arcs orientés. Pour faciliter le processus de génération nous avons défini une bibliothèque d'éléments de Grafcet qui contient les éléments nécessaires pour la construction d'un Grafcet. Pour définir cette bibliothèque, nous nous sommes basés sur la structure d'un Grafcet (Figure 60). L'objectif est de pouvoir utiliser cette bibliothèque pour générer dans un premier temps, un Grafcet générique de fonction qui respecte cette structure et qui est un ensemble d'arcs orientés et d'étapes sans action ni transition associée. Les différentes actions-transitions sont ensuite générées dans le langage ST (Structured Text), puis associées aux étapes dans le Grafcet générique de fonction. En effet, la génération du code de commande est précédée de la génération du code d'action-transition qui est exploité par le modèle générique de grafcet afin d'être adapté à chaque fonction. Ce code est généré à partir des données (configurations généralisées) issues de la généralisation. Le code Action-Transition est un programme qui est appelé aux différentes étapes du Grafcet pour exécuter l'action ou la transition selon l'étape à laquelle il est appelé. L'ordre d'appel de ce code au niveau du Grafcet dépend fortement des séquences d'actions élémentaires (par exemple, ouverture des vannes avant les pompes) effectuées lors de l'enregistrement des exemples. Ces séquences enregistrées devront être généralisées et utilisées dans la génération du code de commande.

3.6 La phase de correction de présentation

Notre démarche est complétée par une phase de correction de présentation au cours de laquelle l'ergonome et l'utilisateur expert (expert métier) travaillent ensemble pour vérifier,

corriger (s'il y a lieu) puis valider la présentation des interfaces de commande de haut-niveau générées. Il s'agit d'offrir la possibilité aux concepteurs de modifier ou non les widgets proposés, s'ils ne leur conviennent pas. Rappelons que le choix des widgets liés aux tâches a été fait avec des règles lors des enregistrements. Ces choix proposés peuvent ne pas être adaptés aux besoins de l'utilisateur final (opérateur en supervision).

Pour permettre la vérification et la validation de la présentation, nous proposons d'intégrer à la démarche de spécification une interface de maquettage qui permettra de modifier, s'il y a besoin, les widgets proposés lors du processus d'enregistrement. L'interface de maquettage est générée à partir de la bibliothèque de widgets et du modèle de tâches de chaque fonction.

L'objectif ici est de proposer à l'expert une interface de maquettage qu'il utilisera pour modifier les différents widgets liés aux tâches du modèle de tâches. Ainsi pour chaque tâche interactive élémentaire, il choisit le widget le mieux adapté à sa réalisation. La modification est réalisée seulement au niveau de la présentation du widget. La modification du dialogue par un expert non-informaticien reste un problème très complexe.

La vérification et la validation de la présentation des interfaces de contrôle-commande générées ainsi que la modification des fausses configurations font partie de l'application des techniques d'EUD (Voir section 2.2) dans notre démarche. Ces techniques mettent l'accent sur la qualité du logiciel que les utilisateurs finaux créent, modifient ou étendent. On retrouve également l'application de ces techniques dans l'application du débogage (détection et correction des erreurs dans les configurations).

4 Bilan sur la spécification fonctionnelle et la génération de systèmes de contrôle-commande

Nous avons proposé dans ce chapitre une approche qui combine les techniques d'EUD et de l'IDM, pour, d'une part, faciliter l'obtention, la vérification, la validation et la correction des spécifications fonctionnelles erronées. D'autre part, notre approche permet de générer la commande de haut-niveau (interface de contrôle et codes de commande) à partir des modèles de tâches et des spécifications fonctionnelles.

Généralement, la construction des commandes de haut-niveau conduit à implémenter manuellement les codes de commande et les interfaces de contrôle. Ces implémentations utilisent les spécifications fonctionnelles, sont longues et fastidieuses en fonction de la complexité du système. En tirant profit des travaux de (Bignon 2012), nous avons proposé une démarche permettant de générer automatiquement les interfaces de contrôle et les codes de commandes pour les commandes de haut-niveau, ceci afin de générer des gains de temps considérables et d'éviter d'éventuelles erreurs.

Notre démarche de spécification fonctionnelle permet de capturer la connaissance de l'expert métier (non-informaticien) sur le système afin d'avoir des spécifications fonctionnelles vérifiées et validées tout en réduisant l'effort de spécification, sans nécessité de former l'expert aux méthodes formelles. En effet, l'introduction des techniques EUD dans un système de spécification donne le pouvoir à l'utilisateur expert d'exprimer naturellement, par utilisation d'une partie de son système habituel, les spécifications fonctionnelles du système qu'il conçoit. Ces spécifications fonctionnelles exemples sont ensuite généralisées, puis utilisées conjointement avec les modèles de tâches, pour générer automatiquement les interfaces de contrôle qui permettent de les vérifier et les valider plus facilement. Les spécifications

fonctionnelles ainsi vérifiées et validées sont ensuite utilisées pour générer les codes de commande.

Avec l'expertise d'un ergonome, l'utilisateur expert peut vérifier et modifier les objets d'interaction contenus dans les interfaces de contrôle générées.

Notre démarche rend l'expert métier et l'ergonome acteurs de la conception. Ils expriment leurs connaissances à travers une IHM d'EGRC (Enregistrement Généralisation Rejeu Correction) intégrée au système d'EUD. Les autres concepteurs (informaticiens et automaticiens), quant à eux, expriment leurs connaissances à travers des éléments de haut-niveau stockés dans des bibliothèques qui sont utilisées par le système d'EBP pour générer les commandes de haut-niveau visibles par la suite sur l'IHM de spécification.

L'utilisation de l'IHM d'EGRC par l'utilisateur expert permet donc de résoudre les problèmes de communication et d'interprétation, ce qui peut réduire considérablement le temps de spécification fonctionnelle d'un système et le risque d'erreur, donc de la réalisation du projet. Cependant la conception de l'IHM d'EGRC intégrant les techniques d'EUD peut être fastidieuse en fonction de la complexité du système de contrôle-commande.

Pour réduire cet effort de conception, il apparaît pertinent d'utiliser les principes de l'IDM qui permettent, par des transformations de modèles, de générer automatiquement l'IHM de spécification.

La méthodologie de génération de l'IHM de spécification, la mise en œuvre des propositions formulées aux chapitres 2 et 3 ainsi que l'obtention du système de contrôle-commande global (système final contenant les commandes élémentaires et de haut-niveau) à partir des spécifications font l'objet du chapitre suivant.

Chapitre 4 : Mise en œuvre des propositions à travers un flot de conception

Ce chapitre détaille la mise en œuvre de nos propositions pour faciliter la conception des modèles de tâches complexes, la spécification fonctionnelle des systèmes complexes, puis l'utilisation de ces deux types de spécifications pour faciliter la conception des systèmes de contrôle-commande.

Pour tenter de répondre à nos problématiques, la première idée consiste à créer manuellement une IHM de spécification qui reprend les éléments du système physique pour être familière à l'expert métier. Cependant, plus le système de contrôle-commande est complexe, plus la conception de cette IHM est fastidieuse. En effet, cette IHM doit être conçue pour chaque système de contrôle-commande à spécifier. Pour éviter cet écueil, nous nous sommes inscrits dans une démarche d'Ingénierie Dirigé par les Modèles (IDM), dans la continuité de travaux précédents présentés dans la section 1.3.3.2 (Chapitre 1, pp. 35) ayant utilisé l'IDM pour la génération d'IHM de contrôle-commande. Ces travaux sous regroupés sous le nom de « Projet Anaxagore » (Bignon 2012). Nous tirons ainsi parti des modèles existants d'Anaxagore pour proposer un flot de conception, présenté sur la Figure 61, qui illustre l'ensemble de notre démarche. Sur la Figure 61 nous mettons en évidence nos contributions liées à la conception des modèles de tâches complexes et à la spécification fonctionnelle des systèmes complexes en combinant l'EUD et l'IDM.

La *première section* de ce chapitre présente le flot de conception proposé, avec une description succincte des opérations qu'il contient.

Afin de comprendre l'implémentation de notre flot de conception, nous définissons dans la *deuxième section* de ce chapitre les outils choisis pour la conception de système de contrôle-commande, ainsi qu'un petit exemple de système.

Les sections suivantes décrivent le détail de l'implémentation des différentes opérations de ce flot.

1. Présentation du flot de conception proposé

Le flot de conception proposé est composé de modèles et d'opérations permettant d'automatiser au maximum l'obtention des modèles finaux opérationnels. Ces automatisations vont ainsi limiter les efforts de conception. Les dix grandes opérations de ce flot sont détaillées dans ce chapitre. Les choix de représentations iconiques permettent aisément de différencier les tâches automatiques de celles qui font intervenir l'expert métier.

La *première opération* porte sur l'opération d'**adaptation des modèles de tâches**. Nous définissons dans cette section la démarche utilisée pour adapter les modèles de tâches des fonctions de haut-niveau de systèmes de contrôle-commandes complexes. Cette opération permet de passer du pattern de tâches aux modèles de tâches.

La *deuxième opération* porte sur l'**adaptation du modèle EGRC**. Les modèles de tâches sont utilisés pour adapter un modèle Enregistreur-Généraliseur-Rejoueur-Correcteur (EGRC) qui intègre les techniques d'EUD. L'adaptation porte également sur la création de propriétés qui permettront d'établir des communications entre le modèle EGRC et les éléments du système à concevoir.

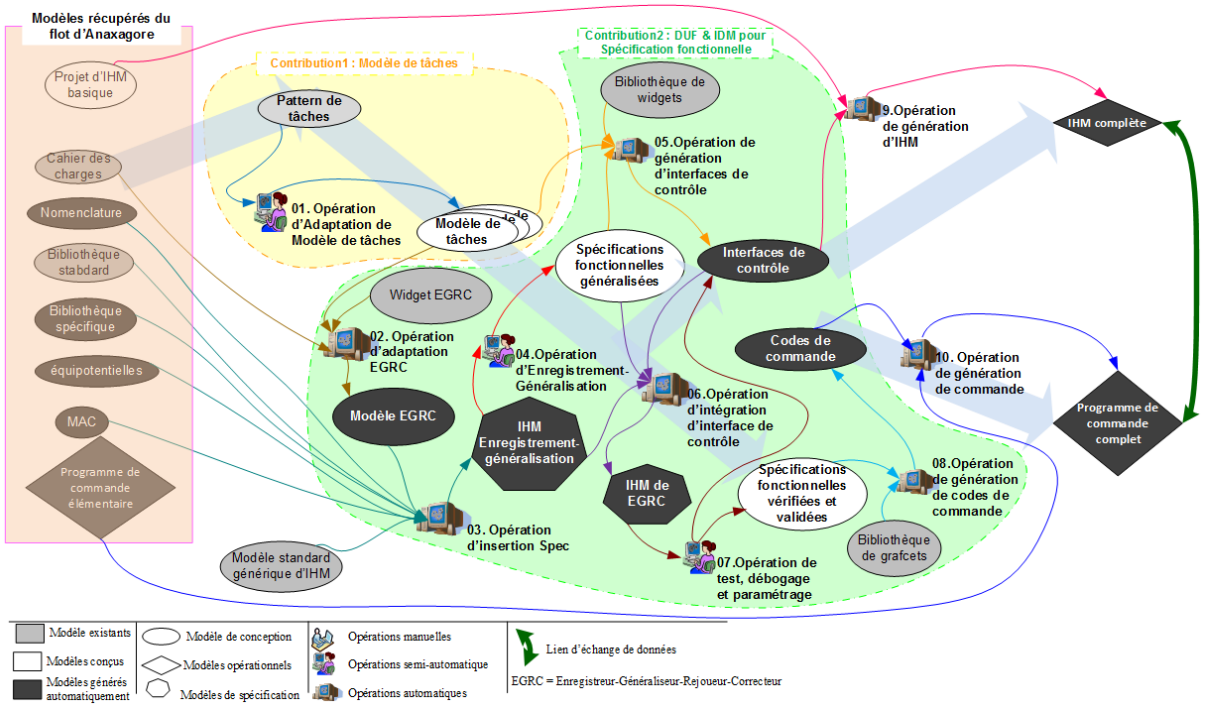


Figure 61 Flot de conception de notre démarche de spécification et de génération de système de contrôle-commande

La *troisième opération* détaille l'opération d'**insertion Spec**. Les informations relatives à la spécification sont extraites des bibliothèques et de la nomenclature, et associées aux équipotentielles, au modèle d'analyse de configuration (MAC), au modèle EGRC et au modèle standard générique d'IHM pour générer une IHM d'enregistrement-généralisation.

L'IHM d'enregistrement-généralisation ainsi générée est ensuite utilisée dans l'opération d'**enregistrement-généralisation** qui est la *quatrième* opération. Cette opération exploite au mieux les connaissances de l'expert en lui offrant une IHM de spécification qui intègre les techniques d'enregistrement-généralisation issues de l'EUD, lui permettant d'obtenir des spécifications fonctionnelles généralisées en construisant des exemples des fonctions attendues.

Les spécifications fonctionnelles généralisées sont utilisées dans l'opération de **génération d'interfaces de contrôle,** la *cinquième opération*, pour produire de façon automatique les interfaces de contrôle contenues dans les commandes de haut-niveau des systèmes de contrôle-commande.

Ces interfaces de contrôle sont par la suite introduites dans l'IHM d'enregistrement-généralisation pour produire une IHM d'EGRC : c'est l'opération de **génération d'intégration d'interfaces de contrôle,** la *sixième* du flot.

L'IHM d'EGRC obtenue permet, d'une part, à l'utilisateur expert de tester, vérifier, corriger et valider les spécifications fonctionnelles du système qu'il conçoit. D'autre part, sur cette IHM, l'expert métier et l'ergonome travaillent ensemble sur les interfaces de contrôle générées pour les rendre plus ergonomiques : c'est l'opération de **test, débogage et paramétrage**, la *septième* du flot. À l'issue de cette opération, on obtient des spécifications fonctionnelles vérifiées et validées.

Les spécifications fonctionnelles vérifiées et validées sont ensuite utilisées dans la *huitième* opération (opération de **génération de codes de commande**), pour générer les codes de

commande créant ainsi le noyau fonctionnel avec lequel les interfaces de contrôle communiqueront.

Ensuite, les interfaces de contrôles générées, puis corrigées, sont utilisées dans l'opération de **génération d'IHM,** *la neuvième*, pour enrichir l'IHM basique. Cette opération vise à obtenir une IHM complète qui permet à la fois une surveillance des éléments du système identifiés sur le P&ID et le contrôle de manière globale par déclenchement de séquences regroupant un ensemble de commandes élémentaires.

Enfin, les codes de commandes générés sont utilisés dans l'opération de **génération de commande**, la *dixième*, pour enrichir le programme de commandes élémentaires. Cette opération vise à la génération du programme de commandes complet qui est lié à l'IHM de supervision complète, par des variables de commande permettant de répondre à la fois aux commandes des éléments du système et aux commandes de haut-niveau.

2. Choix d'outils pour la conception des systèmes de contrôle-commande

Généralement, pour faciliter la conception des systèmes de contrôle-commande, plusieurs logiciels de conception d'application dénommés génériquement SCADA[19] (tels InTouch[20], Panorama E2[21] ou Wonderware[22]) ont été proposés. Ces outils « temps-réel » permettent de visualiser les états physiques ou fonctionnels des équipements et de prendre en charge les fonctions avancées d'un procédé.

2.1. Description des outils utilisés

La description des logiciels utilisés est nécessaire pour la compréhension de l'implémentation de notre démarche. Les outils utilisés pour l'implémentation ont été imposés par le projet Anaxagore, dans lequel s'intègre notre travail. Ils permettent de rendre interprétables les modèles issus de notre démarche, par les outils de notre domaine de travail. Les outils imposés sont Microsoft® Visio, Panorama E2 et Straton, qui ont l'avantage de communiquer sous forme de fichier XML, facilitant ainsi la mise en œuvre des techniques de l'IDM.

Microsoft® Visio est utilisé pour la conception du schéma P&ID (le modèle métier) qui constitue la base de conception des systèmes de contrôle-commande. Généralement l'expression des spécifications fonctionnelles se fait par l'expert métier au travers d'un tableau de configurations non normalisé associé au schéma PID du système qu'il conçoit.

Panorama E2© est un logiciel propriétaire commercialisé par la société CODRA ; il est par exemple utilisé pour superviser le LMJ[23] du CEA. Avec ce logiciel, l'utilisateur peut créer une application de supervision à partir d'objets de base qu'il dispose au sein d'une arborescence construite pour modéliser le procédé à superviser.

Panorama E2 utilise une architecture basée sur un découpage en trois niveaux (Figure 62*)* : le niveau exploitation, le niveau fonctionnel et le niveau procédé. Le *niveau exploitation* est constitué d'outils destinés à l'exploitation de l'installation. Son fonctionnement est lié aux

[19] Supervisory Control and Data Acquisition
[20] www.wonderware.fr
[21] fr.codra.net/
[22] https://www.wonderware.fr/
[23] Laser Mega Joule http://www-lmj.cea.fr/

activités de l'opérateur : il représente l'IHM de l'application. Le *niveau fonctionnel* est le noyau de la supervision. Il assure les fonctions de communication avec les systèmes d'acquisition de données, avec lesquels la supervision communique pour connaître l'état du procédé. Il prend également en charge les traitements étroitement associés au fonctionnement opérationnel du procédé (gestion des alarmes, archivage des données, automatisation…). Enfin, le *niveau procédé* qui se situe hors de Panorama E2 correspond aux systèmes d'acquisition et de contrôle de l'installation supervisée. Les échanges entre Panorama E2 et ce niveau peuvent être mis en œuvre avec un serveur de données de type OPC[24]. Ce niveau assure plusieurs fonctions comme la gestion d'entrées/sorties, l'automatisation, la régulation, et peut être dévolu à un logiciel spécifique, généralement un automate (dans notre cas, nous utilisons le logiciel Straton[2]).

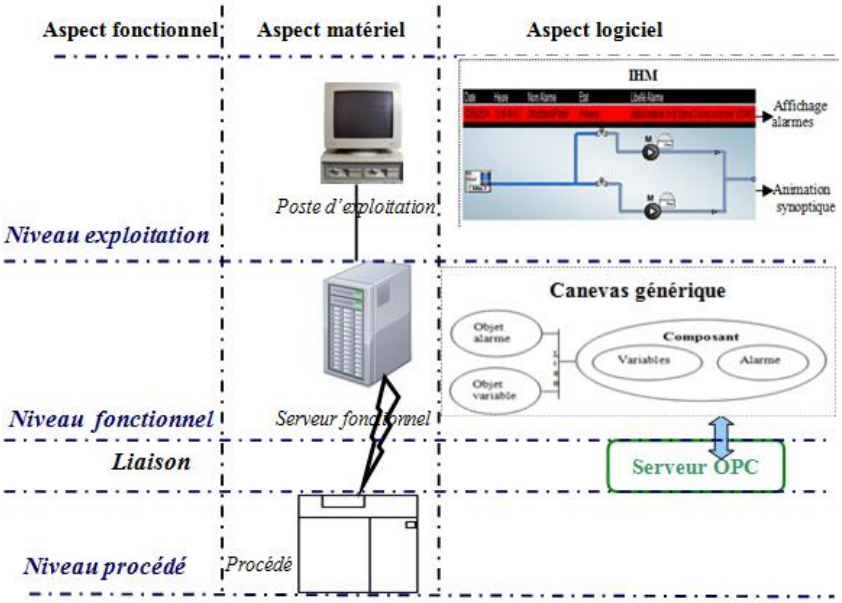


Figure 62 Schéma d'architecture de Panorama E2 (Goubali et al. 2014)

Straton est un logiciel qui prend en charge cinq langages de la norme IEC 61131-3 (SFC, FBD, LD, ST, IL), norme industrielle standardisant la programmation des automates. Il dispose de plusieurs outils permettant le contrôle et la surveillance des applications. De plus, Straton permet de construire des automates programmables sous forme de machine virtuelle exécutable sur ordinateur, sans nécessiter une machine physique supplémentaire.

Par ailleurs, pour la modélisation de l'activité de l'opérateur en supervision, nous avons choisi dans cette thèse, d'utiliser la notation K-MAD. Le choix de cette notation est justifié par le fait que l'approche proposée pour faciliter l'adaptation des patterns de tâches s'appuie sur l'outil *Prototask,* lui-même inclus dans le projet K-MAD.

2.2. Système illustrant la démarche

Enfin, pour la compréhension des différentes opérations, nous décrivons ici un petit système de transfert d'eau douce, représenté le P&ID de la Figure 63, réalisé avec Microsoft Visio©. Ce système permet de transférer de l'eau d'une soute (St1) vers une soute (St2) via l'un ou l'autre des groupes hydrophores (H1 et H2). Un groupe hydrophore est un dispositif qui permet de

[24] www.copalp.com

maintenir la pression dans le réseau. Chaque groupe est isolé de la soute St1 par une vanne motorisée à deux voies (par exemple V2VM01, et V2VM02).

Chaque soute est instrumentée par un indicateur de niveau mécanique (LI0001 pour St1 et LI0002 pour St2, sur la Figure 63), un transmetteur de niveau (LT0001 pour St1 et LT0002 pour St2, sur la Figure 63) et un détecteur de niveau (LS0001 pour St1 et LS0002 pour St2, sur la Figure 63).

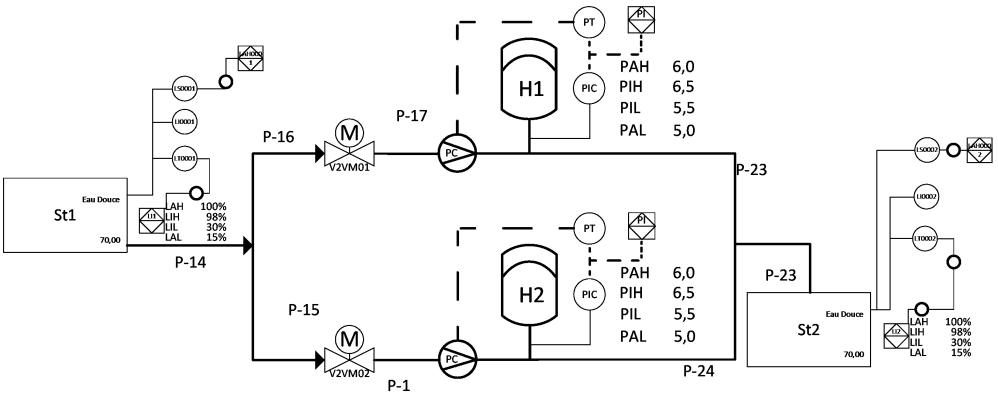


Figure 63 Exemple de P&ID

3. Opération d'adaptation des modèles de tâches

L'opération d'adaptation des modèles de tâches (Figure 64) vise à exploiter le pattern de tâches (Annexe 2) conçu au chapitre 2 à partir du cahier de charges, pour obtenir les modèles des tâches de supervision adaptées aux différentes fonctions de haut-niveau du système à concevoir. Cette opération nous permet de faciliter la conception des modèles de tâches complexes en plaçant l'expert métier au centre de la conception de ces modèles. Le processus d'adaptation est réalisé conjointement avec la simulation du pattern de tâches. Pour ce faire nous proposons un outil proche du simulateur de tâches Prototask (Lachaume et al. 2012) que nous avons appelé Prototask Editor. Avant de définir la transformation mise en œuvre dans l'outil Prototask Editor, nous définissons le concept de cahier de charges et de pattern de tâches.

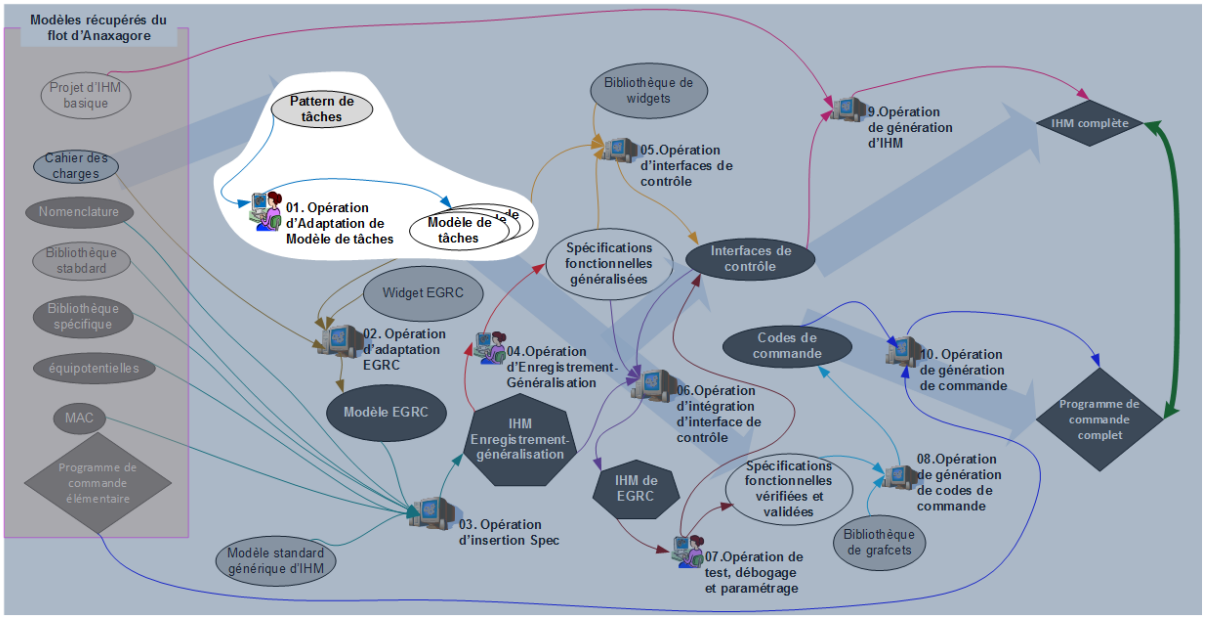


Figure 64 Opération d'adaptation de modèle de tâches

3.1. Les modèles de l'opération d'adaptation des modèles de tâches

3.1.1. Le cahier des charges

Le **cahier des charges** regroupe l'ensemble des exigences sur lesquelles les experts doivent se baser pour concevoir un système. Il constitue une base pour expliquer les besoins aux différents acteurs afin de s'assurer que tous les intervenants le respectent pour la conception du système. De plus, c'est une base fondamentale de communication entre les différents acteurs. Cependant, le cahier des charges n'est pas formalisé (Bignon 2012) et peut être modifié en fonction de l'évolution des besoins du client. La forme de ce document dépend du type et du domaine d'activité principal concerné par le projet.

3.1.2. Le pattern de tâches

En s'appuyant sur l'état de l'art et les retours d'expériences industrielles, nous avons défini un pattern de tâches dans le Chapitre 2 pour décrire les activités des opérateurs en supervision. Ce pattern a été défini avec le langage K-MAD. Il peut être défini avec une autre notation ou être modifié en fonction du système à construire.

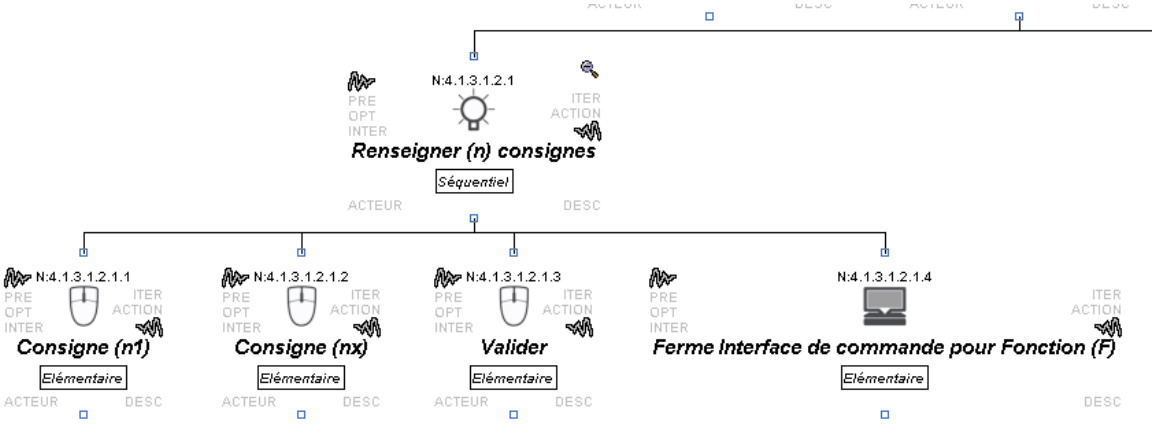


Figure 65 Extrait du pattern de tâches pour décrire la fonction de transfert entre deux soutes

Les patterns de tâches tels que nous les avons définis sont des modèles de tâches génériques. À la différence de ces derniers, les patterns de tâches doivent contenir des variables qui peuvent être adaptées à différents contextes d'utilisation, permettant leur spécialisation en modèles de tâches spécifiques. Pour que le pattern soit le plus générique et le plus flexible possible, la description de la structure hiérarchique des tâches doit comporter au moins un composant variable (par exemple consigne (n1) sur Figure 65). La généricité dans le pattern porte sur les noms des tâches. Les noms des tâches qui sont variables sont ceux qui changent de la description d'une fonction à une autre.

3.2. La transformation de l'opération : simulation et adaptation

Le concepteur (expert métier) doit pouvoir instancier et adapter le pattern de tâches au contexte d'utilisation (aux spécificités de la fonction dont on veut décrire le modèle de tâches). Pour cela, il doit charger le pattern de tâches dans l'outil de modélisation KMADe. L'idée, c'est de ne pas faire éditer un modèle par l'expert, mais de lui faire modifier le pattern tout en réalisant son scénario. La modification porte seulement sur les noms des tâches, car il serait difficile de faire modifier les opérateurs temporels contenus dans le pattern des tâches, par des experts métiers qui n'ont aucune connaissance des notations de tâches. Cependant, ces opérateurs temporels sont utilisés pour simuler le pattern de tâches, suivant les principes de base du simulateur de tâches Prototask détaillés dans (Lachaume et al. 2012).

L'instanciation et l'adaptation s'effectue avec l'outil *Prototask Editor* qui contient en plus des principes de base du simulateur Prototask, des modules permettant au concepteur de modifier le nom des tâches, de rajouter ou supprimer une tâche.

Quatre méthodes sont implémentées pour permettre l'adaptation du pattern de tâches :

- La méthode **DeleteTask** permet de supprimer la tâche sélectionnée et tous les attributs et classes associés dans le fichier XML du Template de tâches. Elle identifie sa tâche mère et modifie l'attribut *id_task_subtasks_list* en y supprimant celle identique à l'*idkmad* de la tâche supprimée.
- La méthode **CheckMotherTask** s'exécute à chaque fois que l'utilisateur supprime une tâche. Elle permet de vérifier le nombre d'*idkmad* dans la liste *id_task_subtasks_list* de la tâche mère. Si ce nombre est inférieur ou égal à 1, un algorithme s'exécute pour supprimer la dernière tâche fille, modifier le nom et la décomposition de la tâche mère. Cet algorithme met la décomposition de la tâche mère à « élémentaire » puisqu'il n'a plus de tâche fille.
- La méthode **ModifyTaskName** permet de remplacer dans le Template de tâches, le nom générique de la tâche sélectionnée par un nom « réel » de tâche.
- La méthode **AddElementaryTask** permet d'ajouter une nouvelle sous-tâche à l'arbre de tâches. Elle intervient au niveau le plus bas de l'arbre de tâches (au niveau des tâches élémentaires). Elle s'exécute lorsque le nombre de sous-tâches élémentaires proposé dans le Template est insuffisant pour détailler la tâche mère.

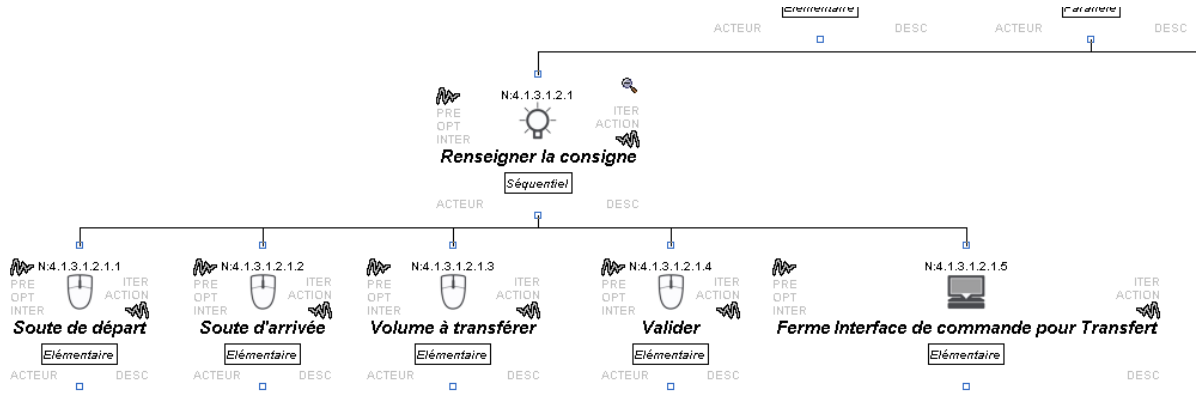


Figure 66 Extrait de modèle de tâches adapté à la fonction Transfert

À l'issue de cette opération nous obtenons des modèles de tâches (Figure 66) de fonctions de haut-niveau vérifiés et validés. Les modèles de tâches ainsi obtenus contiennent des informations nécessaires pour la spécification des fonctions du système, qui doivent maintenant être exprimées par le concepteur.

4. Opération d'adaptation EGRC

Pour faciliter le processus de spécification fonctionnelle proposé, basé sur l'EUD, précisément lors de la démonstration d'exemples de spécification, l'utilisateur expert (expert métier) est guidé pour effectuer les actions nécessaires à la bonne description de l'exemple. Ces actions sont décrites dans les modèles de tâches des fonctions de haut-niveau. Ces modèles sont donc utilisés pour adapter le composant Enregistreur- Généraliseur- Rejoueur- Correcteur (EGRC), qui au départ ne contient aucune information concernant le système à concevoir. L'opération d'adaptation EGRC (Figure 67), la deuxième de notre flot, vise à obtenir le modèle EGRC à

partir du widget EGRC. Nous présentons d'abord les concepts de composant EGRC et de modèle EGRC avant de définir les différentes transformations utilisées dans cette opération.

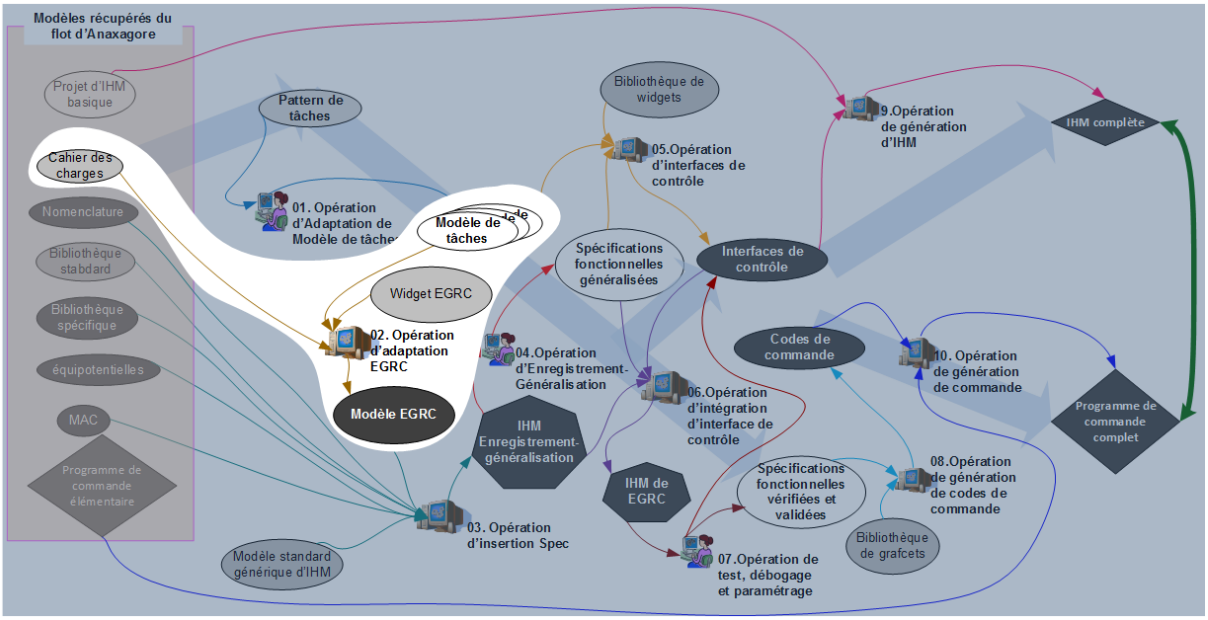


Figure 67 Opération d'adaptation EGRC

4.1. Les modèles de l'opération d'adaptation EGRC

4.1.1. Le composant Enregistreur – Généraliseur – Rejoueur – Correcteur (EGRC)

Les techniques d'EUD sont mises en œuvre à travers un widget EGRC. Ce composant est conçu sous le logiciel de type SCADA Panorama E2 (Goubali et al. 2014) sous forme de vues. Ces vues présentent un enregistreur, un rejoueur et un Correcteur (Figure 68).

Figure 68 Enregistreur/Généarlisaur - Rejoueur/Correcteur (de gauche à droite)

L'*enregistreur* permet de mettre le système en mode enregistrement et de mémoriser toutes les actions de l'expert sur le système. Il est utilisé pour l'enregistrement des exemples de séquences d'exécution pour toutes les fonctions du système à concevoir. Le système effectue une généralisation des enregistrements pour générer un programme exécutable dans un contexte différent. Pour ce faire, le composant EGRC intègre un module de généralisation exploitant la théorie des graphes. Le *rejoueur* permet de rejouer les exemples enregistrés ainsi que ses variantes. Le *correcteur* intègre l'une des techniques de l'EUD et permet à l'ergonome de corriger l'apparence des interfaces de contrôle générées pour les rendre plus ergonomiques. Le caractère générique de ce composant permet son intégration à tout système de supervision. Cependant un paramétrage est nécessaire avec les informations issues des modèles de tâches des fonctions du système à concevoir.

4.1.2. Le modèle EGRC

L'apparence de ce modèle reste identique à celle du composant EGRC (Figure 68). Le modèle EGRC contient en plus de toutes les fonctionnalités du composant EGR, des informations spécifiques au système à concevoir permettant de guider l'utilisateur expert dans les

différentes phases de spécification. Ces informations proviennent du cahier de charges et des modèles de tâches précédemment obtenus.

4.2. Les Transformations de l'opération d'adaptation EGRC

Pour obtenir le modèle EGRC, deux étapes (Figure 69) ont été mise en œuvre : identification de la liste des fonctions du système à concevoir et introduction des informations issues des modèles de tâches et du cahier des charges dans le modèle EGRC générique.

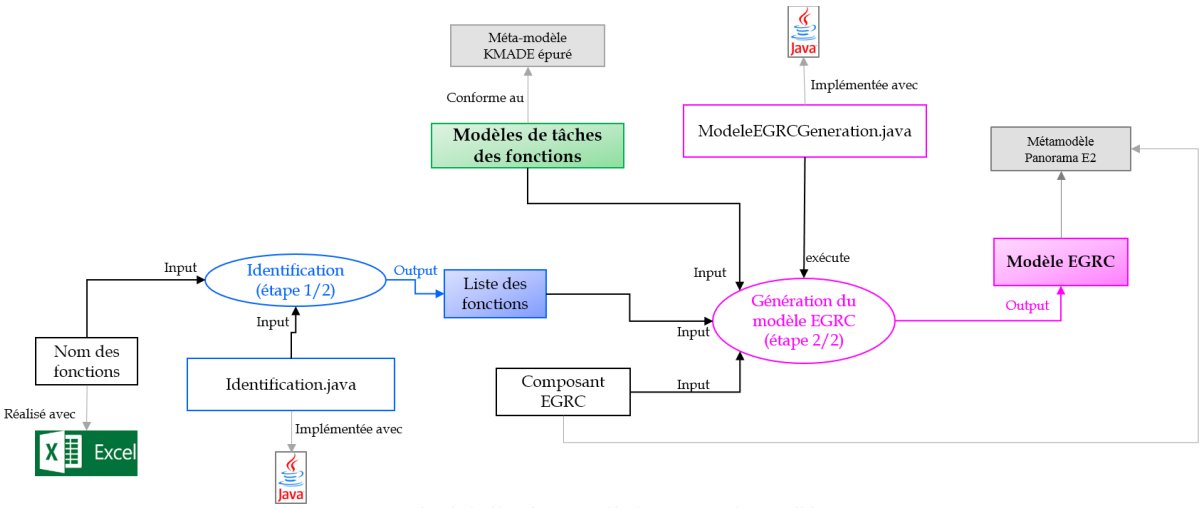


Figure 69 Détail de l'opération d'adaptation du modèle EGRC

4.2.1. Identification de la liste des fonctions

La phase d'identification permet de récupérer la liste des fonctions que doit réaliser le système à concevoir. Deux méthodes sont définies pour réaliser cette phase :

- La méthode **CreateFile** crée le fichier *ListeDesFonctions.txt* dans lequel sera stockée la liste des fonctions à spécifier.
- La méthode **GenerateFunctionList** génère la liste des fonctions à partir d'un fichier excel dans lequel le concepteur (expert métier) a renseigné le nom des fonctions en se basant sur le cahier des charges.

4.2.2. Génération du modèle EGRC

La génération du modèle EGRC vise à introduire les informations issues des modèles de tâches épurés et la liste des fonctions dans le widget EGRC. La Figure 70 présente la structure d'un modèle de tâches épuré. Nous définissons des règles pour ne récupérer que les attributs « taskname » correspondant aux tâches interactives et élémentaires contenues dans l'arbre de tâches « Renseigner & Afficher ». Nous récupérons également l'ordre de réalisation des tâches dans cet arbre de tâches pour pouvoir guider l'expert à réaliser les tâches suivant le même ordre lors de l'enregistrement des spécifications.

Au cours de l'enregistrement des spécifications exemples, le concepteur doit renseigner les conditions qui conduiront le système à arrêter la fonction. La réalisation de cette étape de spécification consiste à comparer les données du système avec les conditions décrites dans le modèle de tâches. Ces informations relatives aux conditions existantes dans les modèles de tâches permettent également d'adapter le widget EGRC. Ainsi, l'adaptation s'intéresse également aux tâches systèmes élémentaires qui décrivent ces conditions.

```
<task classkmad="tache.Task" idtaskpoint="K124" id-task-subtasks-list="K128 K36 K145 K76 " idkmad="K125">
    <taskname>Renseigner consigne </taskname>
    <tasknumero>1.3.1.2.1</tasknumero>
    <taskobservation>Choisir la soute de départ d'arrivée  le volume, puis valider.</task-observation>
    <taskexecutant>ABS</taskexecutant>
    <taskmodality>COGN</taskmodality>
    <taskoptional>false</taskoptional>
    <taskinterruptible>false</taskinterruptible>
    <taskdecomposition>SEQ</taskdecomposition>
    <taskprecondition>true</taskprecondition>
    <taskeffetsdebord>void</taskeffetsdebord>
    <taskiteration>[1]</taskiteration>
    <point classkmad="tache.Point" idkmad="K124">
    <task classkmad="tache.Task" idtaskpoint="K127" idkmad="K128">
        <taskname>Soute de départ</taskname>
        <tasknumero>1.3.1.2.1.1</tasknumero>
        <taskobservation>Choisir la soute de départ</taskobservation>
        <taskexecutant>INT</taskexecutant>
        <taskmodality>COGN</taskmodality>
        <taskoptional>false</taskoptional>
        <taskinterruptible>false</task-interruptible>
        <taskdecomposition>LEAF</taskdecomposition>
        <taskprecondition>true</taskprecondition>
        <taskeffetsdebord>void</taskeffetsdebord>
        <taskiteration>[1]</taskiteration>
        <point classkmad="tache.Point" idkmad="K127">
    </task>
    <task classkmad="tache.Task" idtaskpoint="K35" idkmad="K36">
        <taskname>Soute d'arrivée</taskname>
        <tasknumero>1.3.1.2.1.2</tasknumero>
        <taskobservation>Choisir la soute d'arrivée</taskobservation>
        <taskexecutant>INT</taskexecutant>
```

Figure 70 Extrait de la structure du modèle de tâches

La mise à jour du modèle EGRC est implémentée au moyen d'une applet java. Trois méthodes sont définies :

- La méthode **CreateVarCondition** permet de créer des variables qui sont associées aux conditions d'arrêt identifiées et extraites des modèles de tâches épurés. Ces variables vont permettre l'utilisation de ces conditions lors de la spécification fonctionnelle.
- La méthode **ModifyFunctScript** permet de modifier certains scripts fonctionnels du modèle ER en fonction des variables créées par les quatre premières méthodes.
- La méthode **ModifyExploitScript** permet de modifier certains scripts d'exploitation du modèle ER en fonction des variables créées par les quatre premières méthodes.

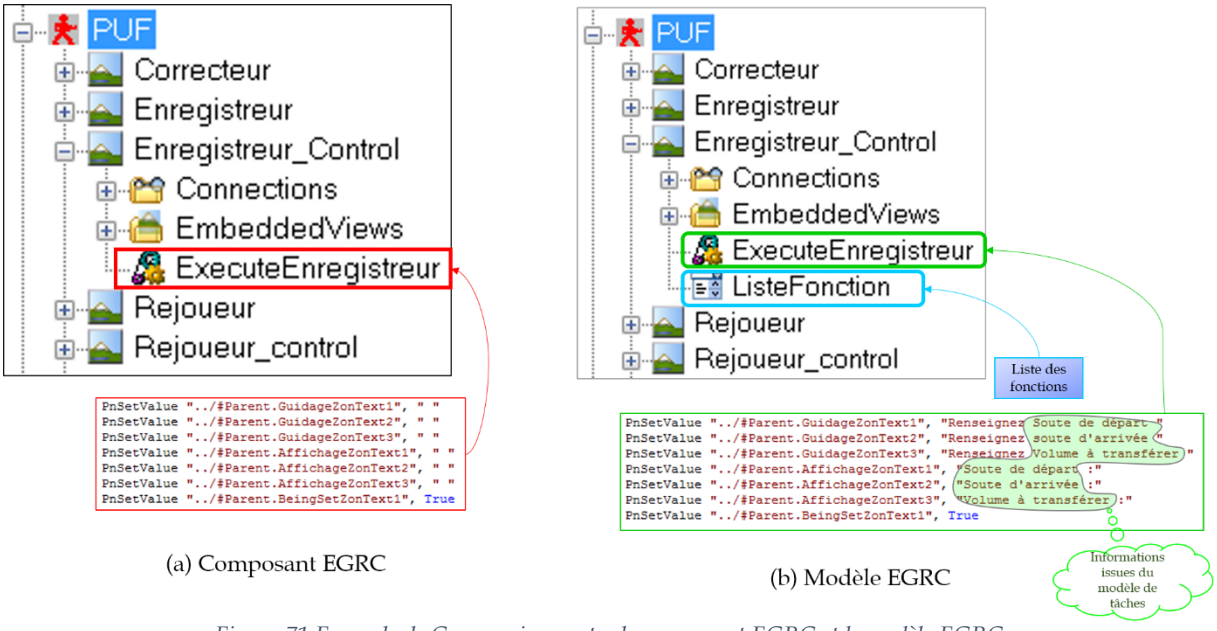


Figure 71 Exemple de Comparaison entre le composant EGRC et le modèle EGRC

À l'issue de cette opération, on obtient un modèle EGRC (Figure 71b) spécifique au système à concevoir et qui peut être utilisé pour la spécification fonctionnelle. La Figure 71 montre un extrait qui illustre la différence entre ce modèle et le composant EGRC. Sur cette figure, il n'y a aucune liste de fonction associée au composant EGRC. De plus, un extrait de script de ce composant montre qu'il est générique. Contrairement à ce composant, le modèle EGRC contient une liste des fonctions du système pour lequel on l'a adapté. Son script est également enrichi et intègre, désormais, des informations issues du modèle de tâches (Figure 71b).

Pour la mise en œuvre de la spécification fonctionnelle, le modèle EGRC est inséré dans une IHM de spécification qui est générée au cours de la prochaine opération.

5. Opération d'insertion Spec

L'opération **d'insertion Spec** (Figure 72) conduit à la génération d'une IHM de spécification qui servira à la spécification des fonctions du système. Elle contient le modèle EGRC et les éléments du système identifiés sur le schéma P&ID qui décrit le système à concevoir.

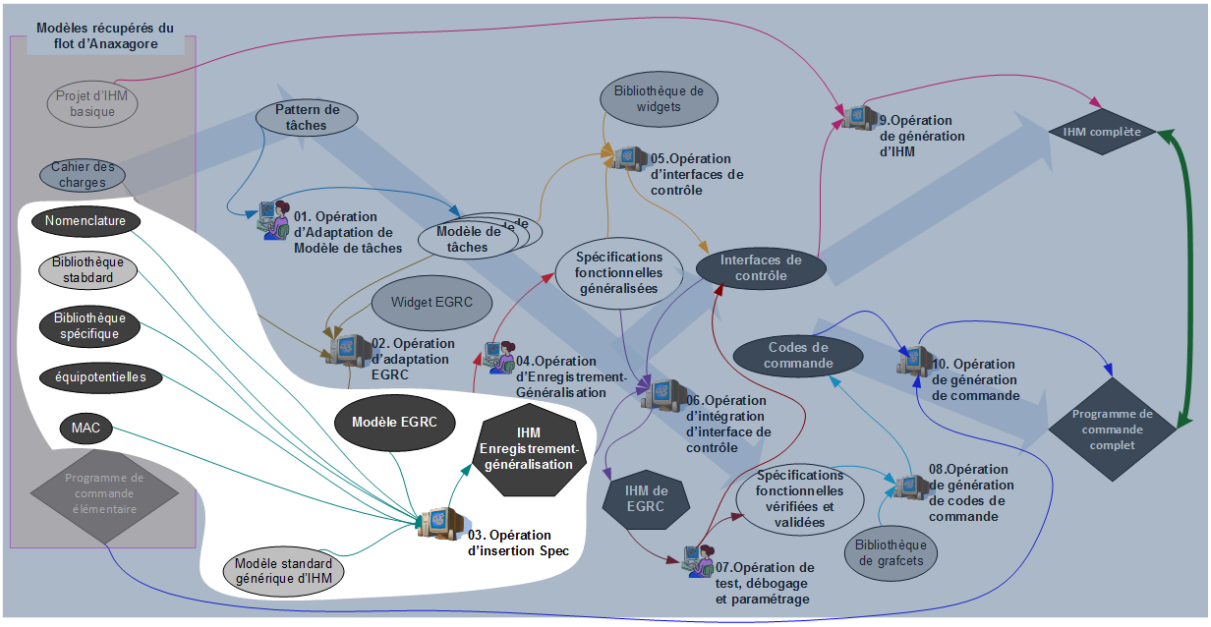


Figure 72 Opération d'insertion spec

L'opération d'insertion Spec consiste à compléter le modèle standard générique d'IHM par les symboles de spécification référencés dans la nomenclature. Les informations de la nomenclature servent à l'instanciation des vues de spécification stockées dans la bibliothèque (*VSpec* sur Figure 73). Une fois instanciées, ces vues sont insérées dans le modèle standard générique, positionnées à l'aide des coordonnées (Coord sur la Figure 76) et reliées grâce aux connexions (Bond sur Figure 76). Nous présentons dans cette section les différents modèles et transformations utilisées pour l'implémentation de cette opération.

5.1. Les bibliothèques

Deux types de bibliothèques sont utilisés dans notre démarche : la bibliothèque standard qui contient les éléments standardisés d'un système et la bibliothèque spécifique qui regroupe les éléments adaptables.

5.1.1. La bibliothèque standard

Les éléments standardisés sont regroupés dans une bibliothèque (Figure 73). Chaque élément comporte des informations organisées en autant de « vues » qu'il y a de concepteurs aux profils différents car chaque expert intervenant sur le projet concentre ses efforts sur la conception des vues relatives à son domaine. Toutes les vues représentant l'ensemble des éléments relatifs aux unités constitutives du système sont liées entre elles par la bibliothèque. La bibliothèque met donc en relation les différentes vues d'un élément standard ; elle regroupe pour chaque vue les aspects relatifs à la commande (*vCmd)*, à la supervision (*vSup*) ainsi que sa représentation sur le synoptique (*vSyn)*. Un élément est rattaché à ses vues par les attributs *L_vSyn* (relatif au synoptique), *L_vSup* (relatif à la supervision) et *L_vCmd* (relatif à la commande) et *L_MAC* (relatif au Modèle d'analyse de Configuration) qui sont des URI[25]. Il est caractérisé par les attributs *family*, *designation* et *reference*. Enfin un élément peut avoir des interfaces (Interface sur la Figure 73).

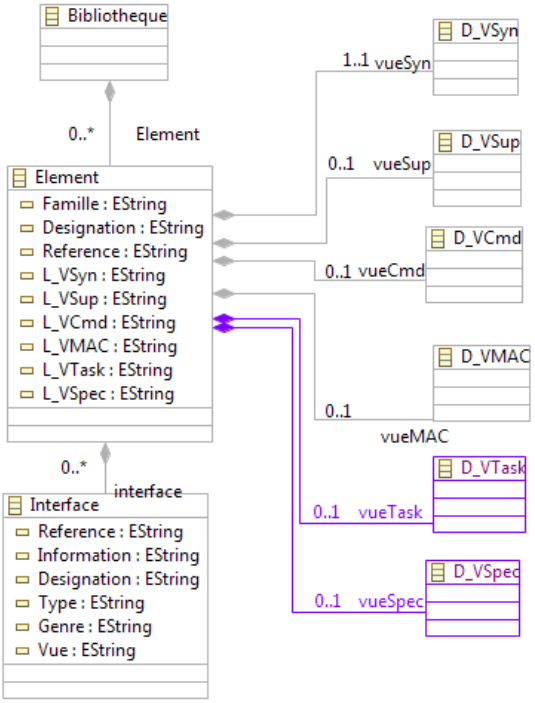


Figure 73 Méta-modèle de la bibliothèque d'Anaxagore

Du fait de la finalité opérationnelle, des outils spécialisés sont utilisés pour créer les vues des éléments. Ainsi la vue de supervision d'un élément est, dans notre cas, implémentée sur Panorama E2. De la même manière, nous utilisons l'environnement de programmation d'automate Straton pour créer la vue de commande. Une vue de commande est en fait un bloc fonctionnel au sens de la norme IEC 61131 (IEC 2003).

Nous avons repris des travaux précédents l'utilisation d'une bibliothèque d'éléments standard pour la génération automatique d'interface. Afin d'intégrer des étapes de spécifications centrée utilisateur dans la démarche existante, nous avons complété la bibliothèque par une vue tâche (*vTask*) qui contient la description du fonctionnement d'un élément du point de vue

[25] Uniform Ressource Identifier, il s'agit dans notre cas d'une chaîne de texte représentant le chemin d'accès à une vue

de l'utilisateur, sous la forme d'un modèle de tâches, ainsi que d'une vue de spécification (*vSpec*) relatif à la spécification fonctionnelle (Figure 73).

La Figure 74 présente le modèle de tâches d'une vanne à deux voies motorisée. Ce modèle décrit les actions que l'opérateur en supervision peut effectuer pour agir sur une vanne. Il peut soit ouvrir une vanne, si elle n'est pas ouverte, soit fermer une vanne qui est ouverte. L'ouverture et la fermeture d'un élément s'effectuent suivant une séquence d'actions bien définie décrite dans le modèle de tâches de l'élément. Ce modèle servira de base pour d'une part améliorer la vue de supervision déjà existante dans la bibliothèque, et d'autre part, construire la vue de spécification de l'élément.

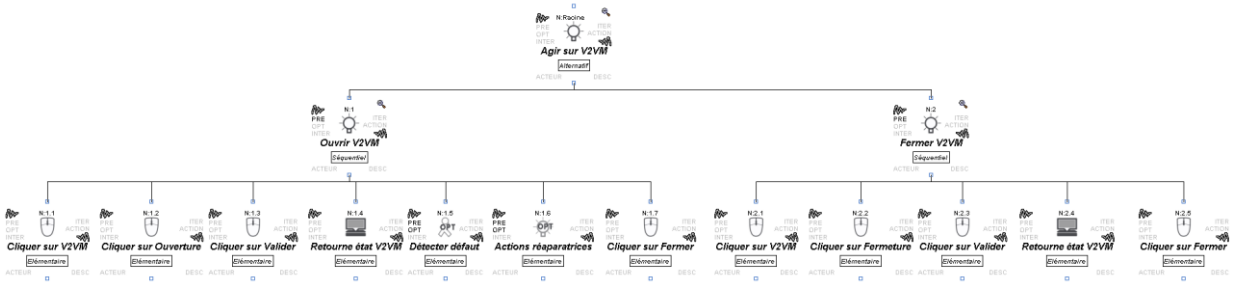


Figure 74 Vtask d'une vanne V2VM

```
'si le système est en mode enregistrement
If PnGetValue("../#Parent.Recording") = True Then
    'on récupère le nom du composant sur lequel l'utilisateur agit.
    p = PnGetValue("../#Parent.Repere")
    'on récupère la valeur de la variable de supervision qui sera envoyée à la commande.
    p1 = PnGetValue("../#Parent.Ctrl0")
    'on écrit dans le fichier de configuration l'état du composant.
    F1.WriteLine("Ctrl0_" & p)
    'on écrit dans un autre fichier le programme correspondant à l'action que fait l'utilisateur.
    F2.write("PnSetValue" & chr(34) & "../#Parent.Ctrl0_" & p & chr(34) & ", ")
    F2.writeline(p1)
End If
```

Figure 75. Extrait du script de la vSpec d'une vanne

Visuellement la vue de spécification est créée avec le même outil que la vue de supervision et par le même expert. La différence entre la vue de supervision et la vue de spécification est que cette dernière est dotée d'un script qui lui permet d'être à l'écoute des actions de l'expert (Figure 75).

5.1.2. La bibliothèque spécifique

La bibliothèque d'éléments contient à l'origine des éléments standardisés extrêmement spécialisés. Tous les éléments de conception d'un système ne peuvent être standardisés. La contrepartie est que ces éléments ne peuvent pas être utilisés pour une fonction détournée. Or, certains éléments d'un système sociotechnique sont si particuliers ou évoluent tellement rapidement qu'il n'y a pas d'intérêt à les standardiser. Pour ces éléments spécifiques, le surcoût engendré par la standardisation rend cette opération contre-productive. Nous avons donc défini dans la bibliothèque un élément générique qui peut être adapté à un besoin spécifique. Cet élément dispose d'une importante généricité, c'est-à-dire, regroupe les fonctionnalités les plus répandues dans les autres éléments de la bibliothèque. Nous proposons donc un élément adaptable qui reprend les caractéristiques des éléments standard de la bibliothèque d'Anaxagore. À partir de cet élément adaptable, nous avons par une opération de spécialisation, obtenu des éléments spécifiques au système à concevoir. Ces éléments sont stockés dans une bibliothèque spécifique.

La **bibliothèque spécifique** contient pour chaque composant spécifique obtenu après adaptation, les aspects relatifs à sa vue de supervision et à sa vue de commande.

5.2. La nomenclature

Le modèle de la **nomenclature** est un modèle généré, à l'issu des travaux de (Bignon 2012), à partir des *bibliothèques* et du modèle de *synoptique (le schéma P&ID du système à concevoir)*. La nomenclature lie les instances listées à partir du synoptique aux vues des éléments correspondant dans les bibliothèques ainsi que les connexions (*Bond*) établies entre les éléments sur le synoptique (Figure 76).

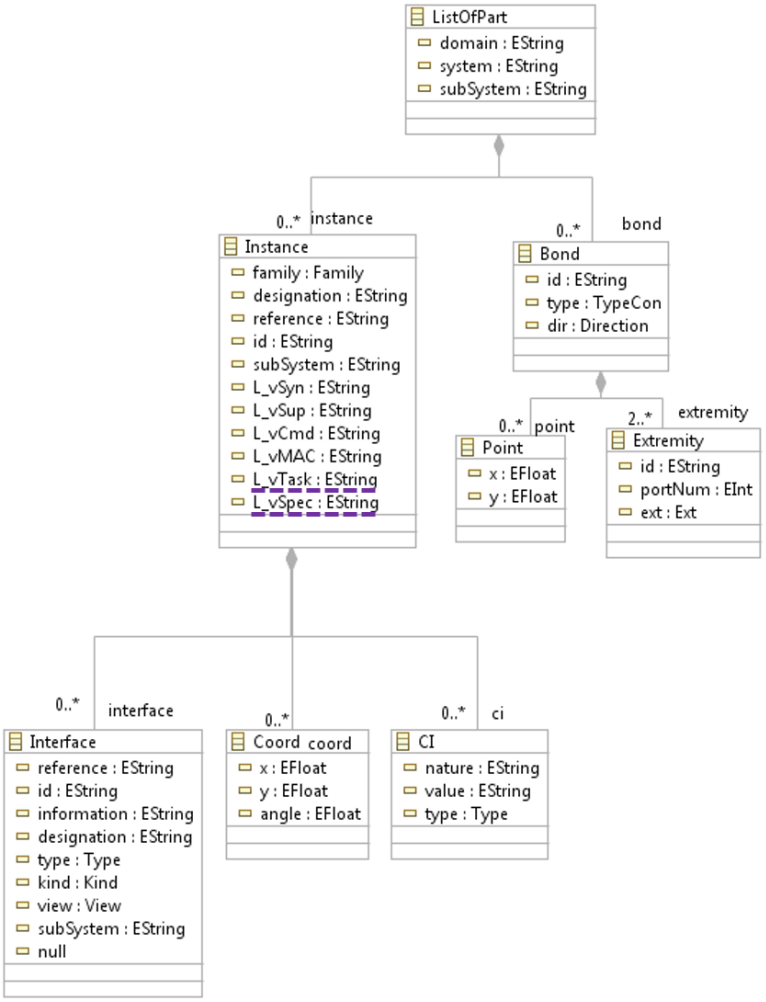


Figure 76 Extension du méta-modèle de la nomenclature

Une instance peut être rattachée à des *Interfaces*, des coordonnées (*coord*) et des informations complémentaires (*CI*). Les interfaces permettent de lister les variables échangées entre la supervision et la commande. Les coordonnées (*coord*) sont conservées de la même manière que sur le synoptique pour pré-positionner les symboles de supervision (vSup) sur les IHM. Les informations complémentaires (*CI* sur Figure 76) sont renseignées lors de la conception du schéma P&ID et permettent d'avoir des précisions (par exemple, seuils de détection des niveaux très bas, bas, haut et très haut) sur certains éléments du système.

Le modèle de nomenclature généré contient toutes les informations nécessaires à la génération d'IHM de contrôle-commande correspondant au synoptique.

```
<instance family="Process" designation="Vanne 2 Voies Motorisee" reference="V2VM" id="V2VM01"
vSyn="V2VM\VSyn\" vSup="V2VM\VSup\" vSpec="V2VM\VSpec\" vTask="V2VM\VTask\" vCmd="V2VM\VCmd\"
vMAC="V2VM\VMAC\" subSystem="EdS">
  <interface reference="V2VM" id="V2VM01" information="CtrlO" designation="Commande Ouvert/Fermer"
  type="BOOL" kind="Output" view="VSpec" subSystem="EdS"/>
  <interface reference="V2VM" id="V2VM01" information="StO" designation="Etat vanne Ouverte"
  type="BOOL" kind="Input" view="VSpec" subSystem="EdS"/>
  <interface reference="V2VM" id="V2VM01" information="StFMat" designation="Defaut Materiel"
  type="BOOL" kind="Input" view="VSpec" subSystem="EdS"/>
  <interface reference="V2VM" id="V2VM01" information="StFCmd" designation="Etat defaut de commande de la vanne"
  type="BOOL" kind="Input" view="VSpec" subSystem="EdS"/>
```

Figure 77 Extrait de la nomenclature après extension

Pour utiliser la nomenclature dans la génération d'IHM de spécification, nous avons complété son méta-modèle par les mêmes vues que celles ajoutées à la bibliothèque (5.1) : une vue tâche (*vTask*) et une vue de spécification (*vSpec*) (Figure 76). On ajoute à la classe instance de la nomenclature les attributs *vTask* et *vSpec* qui contiennent les chemins d'accès aux vues de tâche et de spécification de l'instance d'un élément (Figure 76 et Figure 77).

5.3. Le MAC (Modèle d'Analyse de Configuration)

Le MAC (Modèle d'Analyse de Configuration) (Figure 78) modélise l'architecture du système sous forme de graphes de potentiels et d'un ensemble de contraintes. Ce modèle permet la mise en œuvre d'algorithmes de recherche de flot maximum dans un graphe orienté pour déterminer les configurations. Il est issu des travaux de (Bignon 2012), et est calculé à partir des vMAC de chaque élément de type procédé de la *bibliothèque standard*. La Figure 78 montre le modèle d'analyse de configuration généré à partir du P&ID (Figure 63) et des vMAC (Figure 73).

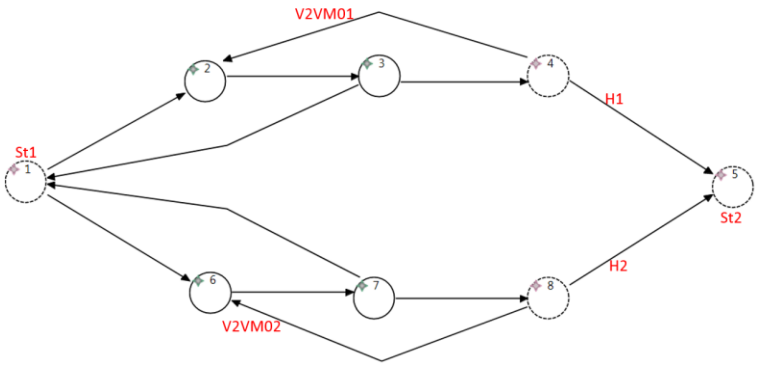


Figure 78. Extrait d'une modèle d'analyse de configuration du système de la Figure 63

5.4. Les équipotentiels

Dans un système de supervision, il est essentiel de rendre visible l'état du système. Dans un système de distribution d'eau par exemple, la visualisation de la mise en eau des circuits est un élément important, et sa traduction par une animation est largement préférable à toute autre forme de visualisation. D'un point de vue général, la dynamique de l'animation nécessaire à la mise en évidence des phénomènes physiques d'un système nécessite une interaction entre les composants graphiques et est souvent programmée « à la main ». On imagine bien la difficulté de programmer dans un logiciel comme Panorama E2 la dynamique entre les composants graphiques nécessaires à la bonne compréhension de l'opérateur. Pour faciliter la gestion de l'animation, *Anaxagore* permet de générer les équations permettant d'animer l'IHM de contrôle-commande à partir des équipotentiels (Bignon 2012). Un équipotentiel est l'ensemble des points où un potentiel prend une même valeur. Toutes les

connexions du système ayant une même extrémité ont une même valeur de potentiel et donc font partie du même équipotentiel.

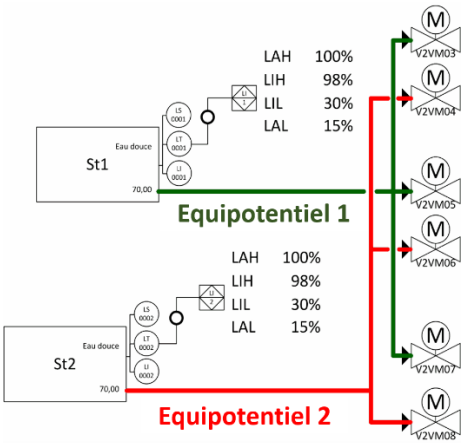


Figure 79 Illustration des équipotentiels (Bignon 2012)

Les équipotentiels (Figure 79) regroupent des zones du système qui ont toutes le même potentiel. Elles permettent de prendre en compte le système dans son ensemble pour générer des équations d'animations de la propagation du fluide. Les équipotentiels permettent, au travers de la liste établie, de générer des équations d'animation de la propagation de fluide dans les symboles de tuyauterie. Les symboles du canevas d'IHM ne sont donc plus indépendants les uns des autres. Cet affichage permet à l'opérateur de visualiser rapidement la configuration du système.

5.5. Le modèle standard générique d'IHM

La Figure 80 illustre la structure hiérarchique simplifiée de notre modèle standard réalisé sous Panorama E2. Cette structure présente les domaines contenus dans ce modèle, qui sont : Plateforme, Navire, Réglages. Le domaine *Plateforme* est lui-même subdivisé en systèmes (*Auxiliaires*, *Propulsion*, *Sécurité*, *Électricité*). Nous nous intéressons dans cette thèse à un sous-système du système auxiliaire, il est défini par un fichier *Unit.cfg* et par un fichier *Canevas.cfg*. Le fichier de configuration *Unit.cfg* définit l'architecture de l'interface et le fichier de configuration du projet *canevas.cfg* définit la structure de l'affichage de l'interface.

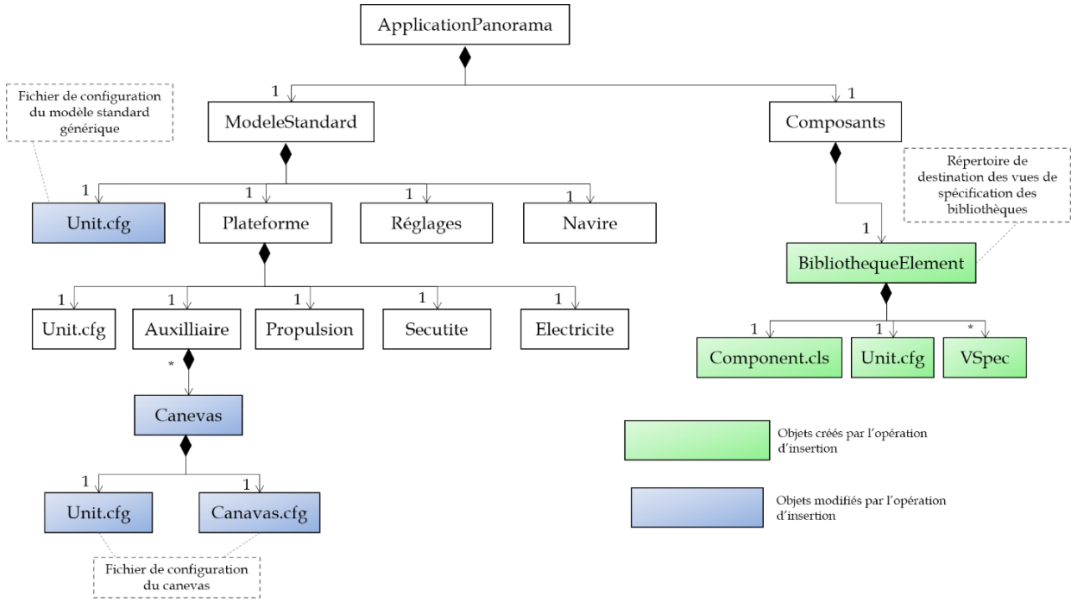


Figure 80 Structure du modèle standard générique sous Panorama E2 (Bignon 2012)

Le modèle standard d'IHM est vu comme un patron de conception puisqu'il impose des règles d'agencement des zones de l'IHM (Bignon 2012).

Le Zoning schématise l'organisation visuelle des principales zones constituant une interface. Il est généralement réalisé au cours de la première phase d'analyse par l'ergonome, où il délimite les zones grossièrement après une série d'entretiens auprès des utilisateurs. À partir des résultats d'entretiens passés avec des utilisateurs expérimentés de l'interface à construire et en se basant sur une étude de zoning de différentes interfaces existantes, un zoning a été réalisé. Ce zoning a permis à l'ergonome de déduire un modèle standard générique qui regroupe les règles de présentation de l'interface à développer. Le modèle standard générique est composé de trois zones : la zone de menu, la zone de commande et la zone de visualisation. La zone de visualisation dispose d'un espace plus grand que les deux autres zones. L'objectif est de pouvoir y intégrer des représentations visuelles claires et détaillées. Chaque zone peut recevoir plusieurs sous-zones en fonction du domaine d'application et des règles métiers liées à ce domaine.

Par exemple dans le cas des systèmes de contrôle-commande embarqués sur un navire, la ***zone de menu*** reçoit les sous-zones alarme, titre (le nom du système représenté sur l'interface), menus de navigation entre les interfaces des différents sous-systèmes du navire (la synthèse du navire, le sous-système de Propulsion, le sous-système de sécurité, d'électricité, l'auxiliaire, le réseau de réglage, etc.). La ***zone de commande*** reçoit les commandes de haut niveau permettant de réaliser les fonctions du système. La ***zone de visualisation*** reçoit la représentation du système sélectionné. Tous les composants du schéma P&ID du système à concevoir sont représentés par leur vue de supervision et agencés conformément au synoptique. On peut également visualiser dans cette zone, la représentation d'une synthèse du système.

5.6. L'IHM d'enregistrement-généralisation

L'**IHM d'Enregistrement-généralisation** (Figure 81) intègre un module de généralisation. Elle est générée automatiquement par une succession d'opérations.

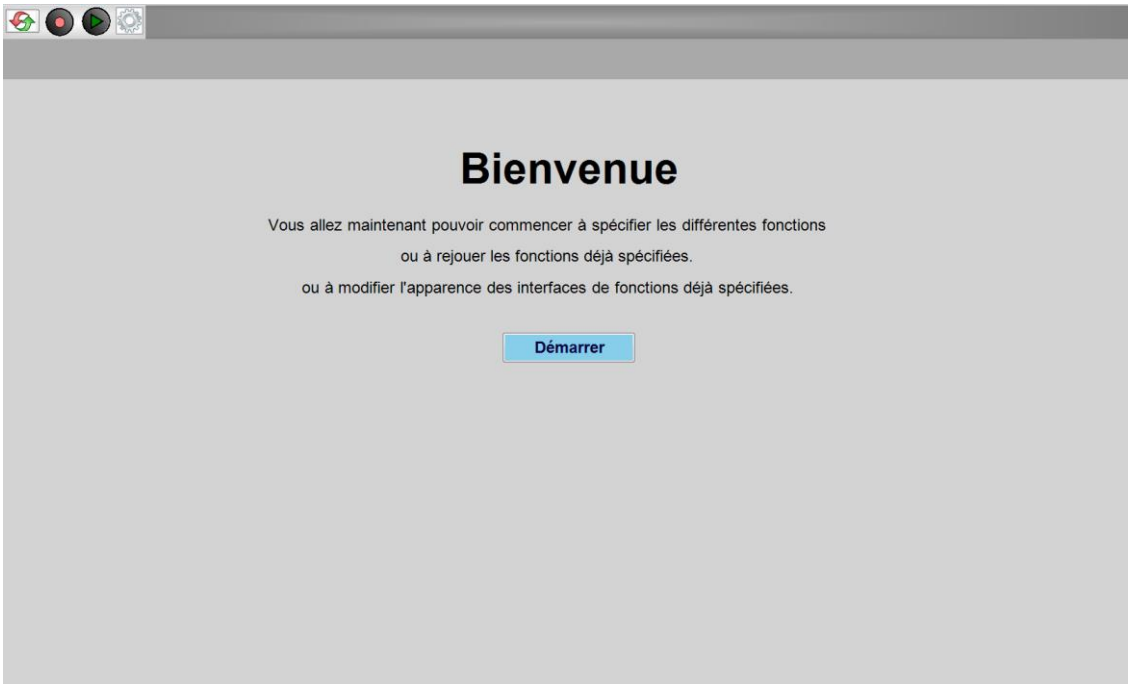


Figure 81 Résultat de l'opération d'insertion Spec – page d'accueil de l'IHM d'Enregistrement-généralisation

C'est une IHM exploitable par Panorama E2, qui offre une représentation graphique du système physique. L'ensemble du module de spécification (interface et module de généralisation) intègre les techniques du EUD pour permettre aux concepteurs (non-informaticiens) d'obtenir les spécifications fonctionnelles de leur système sous une forme directement exploitable pour l'obtention des spécifications fonctionnelles puis des commandes de haut-niveau.

5.7. Les phases de l'opération d'insertion Spec

Le modèle standard générique d'IHM et le « projet d'IHM de spécification » généré sont conformes au méta-modèle de Panorama E2. Le projet d'IHM de spécification comporte une bibliothèque d'éléments regroupant toutes les vues de spécification des composants prédéfinis et deux fichiers de configuration par interface : le premier décrit l'architecture de l'interface, et le second décrit la structure de l'affichage de l'interface. Les transformations réalisées dans l'opération d'insertion Spec se déroulent en quatre phases (Figure 82) : la phase de préparation, la phase de création de l'architecture, la phase de création de la structure et la phase de liaison du composant ER aux éléments du système.

La phase de préparation sans prise en compte du modèle EGRC, la phase de de génération d'architecture et la phase de structure avaient été déjà implémentées dans les travaux de (Bignon 2012), pour la génération d'une IHM de supervision de bas-niveau. Nous avons repris ces trois phases en y intégrant, dès la phase de préparation, le modèle EGRC.

Pendant la phase de préparation, le modèle ER et les éléments disposant d'une vue de spécification, associés à leurs instances dans la nomenclature sont, dans un premier temps, insérés dans la bibliothèque de Panorama. Puis l'utilisation d'une applet Java permet d'identifier les dimensions des symboles afin de les positionner sur l'IHM de spécification conformément au schéma PID. Le modèle EGRC est également positionné au cours de cette phase, sur la zone de menu du modèle standard générique.

Lors de la deuxième phase, les composants de l'IHM de spécification avec leurs dimensions sont instanciés conformément aux instances listées dans la nomenclature (Figure 76). Cette phase a été réalisée avec une transformation ATL détaillée dans (Bignon 2012).

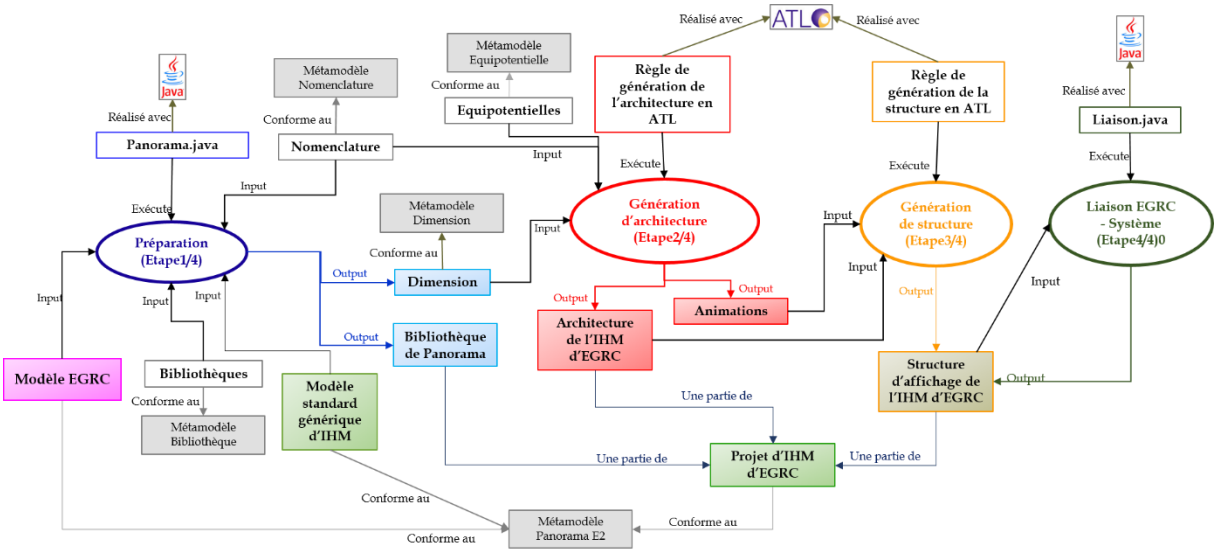


Figure 82 Détail de l'opération d'insertion « Vspec »

La dernière phase permet de créer concrètement l'IHM en détaillant la position des symboles instanciés.

La phase de liaison du modèle EGRC aux éléments du système permet d'établir une liaison entre ce modèle et les autres éléments du système. En effet, le but du modèle EGRC est de communiquer avec les éléments du système à concevoir. Pour cela, il est nécessaire de créer des variables de commande qui seront directement liées aux éléments du système. Cette création de variables exploite les variables de chaque élément disposant d'une vue de spécification.

Pour permettre la communication entre les éléments du système et le modèle EGRC, plusieurs types de propriétés doivent être liés au modèle EGRC et entre eux. Deux types d'éléments peuvent être différenciés : les éléments qui peuvent être des points de départ et/ou d'arrivée et tous les autres éléments. Certaines propriétés sont uniquement liées aux éléments de départ/arrivée et d'autres, aux autres éléments du système. Lors de la phase de liaison, ces deux types d'éléments doivent être différenciés. Trois méthodes sont implémentées pour la mise en œuvre de cette phase :

- La méthode **ListElement** parcourt la nomenclature et crée la liste de tous les éléments du système, tout en distinguant les deux types d'éléments précédemment cités.
- La méthode **LinkPropEGRC** parcourt la liste créée par la première méthode identifie les propriétés nécessaires à la communication avec le modèle EGRC, crée et lie ces propriétés à ce modèle. La Figure 83 montre la liaison des propriétés des éléments du système à celles du modèle EGRC, nommé PUF.

```
<OBJECT CLASS="PUF" ID="PUF" MODULE="Bibliotheque">
<PROP ID="sto_v2vm01" SCRIPT="V2VM01.StO"/>
<PROP ID="sto_v2vm02" SCRIPT="V2VM02.StO"/>
<PROP ID="startedelement1" SCRIPT="V2VM01.OpenElement"/>
<PROP ID="startedelement2" SCRIPT="V2VM02.OpenElement"/>
<PROP ID="startedelement21" SCRIPT="H1.StartedElement"/>
<PROP ID="startedelement22" SCRIPT="H2.StartedElement"/>
```

Figure 83 Extrait des liaisons réalisées dans la PUF

- La méthode **LinkPropElement** identifie les propriétés des éléments qui sont contenus dans le modèle EGRC pour les lier aux éléments du système. La Figure 84Figure 83 montre la liaison des propriétés des éléments du système à celles du modèle EGRC, nommé PUF.

```
<OBJECT CLASS="V2VM" ID="V2VM01" MODULE="Bibliotheque">
<PROP ID="nomfonction" SCRIPT="PUF.NomFonction"/>
<PROP ID="recording" SCRIPT="PUF.Recording"/>
<PROP ID="replay" SCRIPT="PUF.Replay"/>
<PROP ID="edit" SCRIPT="PUF.Edit"/>
<PROP ID="ctrlo" SCRIPT="PUF.CtrlO_V2VM01"/>
<PROP ID="sto" SCRIPT="PUF.StO_V2VM01"/>
<PROP ID="repertory" SCRIPT="PUF.Repertory"/>
<PROP ID="recordingparametres" SCRIPT="PUF.RecordingParametres"/>
<PROP ID="parametresrecord" SCRIPT="PUF.ParametresRecord"/>
<PROP ID="recordingpoint" SCRIPT="PUF.RecordingPoint"/>
<PROP ID="pointrecord" SCRIPT="PUF.PointRecord"/>
```

Figure 84 Extrait des liaisons réalisées dans la vanne V2VM01

À l'issue de ces trois phases nous obtenons une IHM d'enregistrement – généralisation interprétable (Figure 81 et Figure 85) sous Panorama E2. Cette IHM contient les symboles issus des vues de supervision, conformément au schéma P&ID de la Figure 63. On y retrouve les soutes (St1 et St2) dont la capacité de 70 m3 chacune correspond aux informations du schéma P&ID.

On retrouve également les connexions entre les différents éléments du système : les vannes (V2VM01 et V2VM02), les pompes (H1 et H2) et les soutes (St1 et St2).

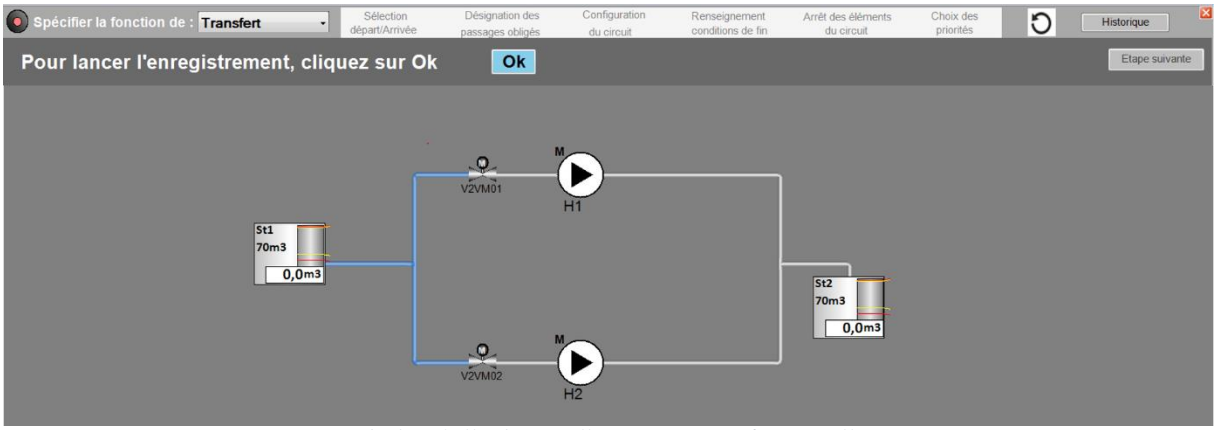


Figure 85 Résultat de l'opération d'insertion Spec –focus sur l'enregistreur

On y retrouve également les différents widgets (Enregistreur, rejoueur, correcteur) du modèle EGRC (Figure 81). La Figure 85 montre l'IHM d'EGRC lorsqu'on clique sur l'enregistreur. On retrouve alors les différentes étapes qui doivent suivies par l'utilisateur expert pour enregistrer des exemples de fonctions.

6. Opération d'enregistrement-généralisation

L'**opération d'enregistrement-généralisation** (Figure 86) vise à utiliser l'IHM précédemment générée pour capturer la connaissance de l'expert (non-informaticien) sur le système à concevoir afin d'avoir des spécifications fonctionnelles généralisées.

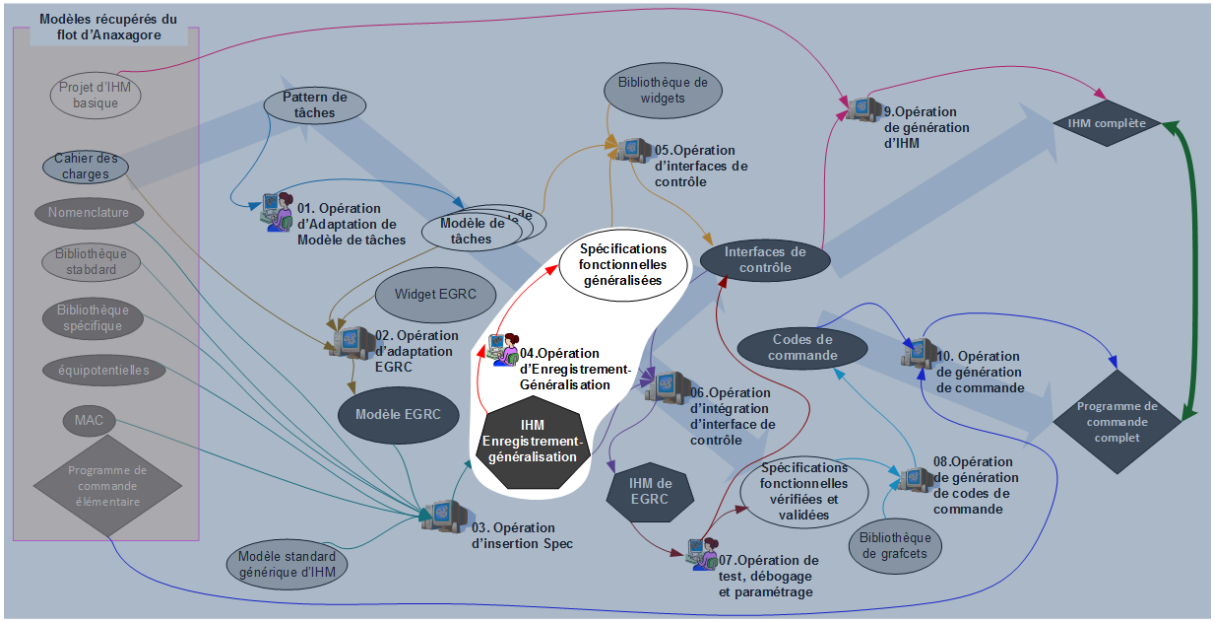


Figure 86 Opération d'enregistrement-généralisation

6.1. Les spécifications fonctionnelles

Pour notre étude, nous considérerons les spécifications fonctionnelles comme les suites d'actions de l'opérateur sur le système, nécessaires à la réalisation d'une fonction avec prise en compte de toutes les possibilités (configurations). Elles décrivent la façon dont un système permet d'atteindre les buts de l'utilisateur et contiennent notamment une liste des principales

fonctions du système (exigences de haut niveau) et des scénarios d'exploitation. Elles permettent de traduire les exigences liées à la réalisation des tâches de l'utilisateur et doivent contenir pour chaque fonction, les informations suivantes :

- La liste des éléments qui peuvent être un point de ***départ*** pour la fonction. Par exemple, pour le système de la Figure 63, l'élément de départ pour la fonction de Transfert est St1.
- La liste des éléments qui peuvent être un point d'***arrivée*** pour la fonction. Par exemple, pour le système de la Figure 63, l'élément d'arrivée pour la fonction de Transfert est St2.
- La liste des éléments indispensables à la réalisation de la fonction. Si nous reprenons l'exemple de la Figure 63, les éléments indispensables sont les hydrophores (H1 et H2 sur Figure 63) qui permettent de pomper l'eau dans la soute St1. Sans la présence de l'un au moins de ces éléments la fonction ne peut se réaliser. Nous appelons ces éléments ***points de passages obligés***.
- Toute la séquence d'actions de l'utilisateur expert sur le système, qui en réalité, un programme que nous appelons ***dialogue intra-widget***.
- La réaction du système par rapport aux actions de l'utilisateur expert, qui est également un programme, que nous appelons ***dialogue intra-fenêtre***.
- Une liste d'objets interactifs proposés à l'opérateur pour effectuer les actions nécessaires au lancement d'une fonction. Nous appelons cette liste modèle d'interaction.
- Toute la séquence d'actions effectuée, dans un ordre précis, sur les éléments du système pour réaliser la fonction. Cette séquence d'actions est sous forme de programme que nous appelons ***séquences***.
- Toutes les ***configurations*** nécessaires à la réalisation de chaque fonction.
- Les priorités correspondantes à chacune des fonctions du système à concevoir.

Toutes ces informations peuvent être recueillies par le système à travers la réalisation d'un exemple. La zone de menu, l'enregistreur contenu dans l'IHM enregistrement-généralisation (Figure 85), est organisée de manière à recueillir toutes ces informations.

6.2. Les phases de l'opération d'enregistrement - généralisation

L'opération d'enregistrement-généralisation se déroule en deux phases : l'enregistrement et la généralisation. Au cours de la première la phase l'utilisateur expert doit utiliser l'IHM générée pour enregistrer des exemples de spécifications fonctionnelles. La deuxième phase part de ces exemples pour proposer les spécifications fonctionnelles de l'ensemble du système.

6.2.1. La phase d'enregistrement

Cette phase est manuelle et vise à obtenir des exemples de spécifications fonctionnelles (Figure 87). Alors que, dans la plupart des systèmes d'EBP, la construction de l'exemple se fait à l'initiative de l'utilisateur, notre démarche s'appuie sur les modèles issus des étapes antérieures, et plus spécifiquement le modèle EGRC. Ceci permet de guider l'expert dans sa démarche, réduisant fortement son besoin de connaissances en programmation. Pour enregistrer des exemples de fonction de haut-niveau, l'utilisateur expert est guidé par modèle EGRC que le système d'enregistrement – généralisation utilise pour déterminer les différentes étapes de l'enregistrement.

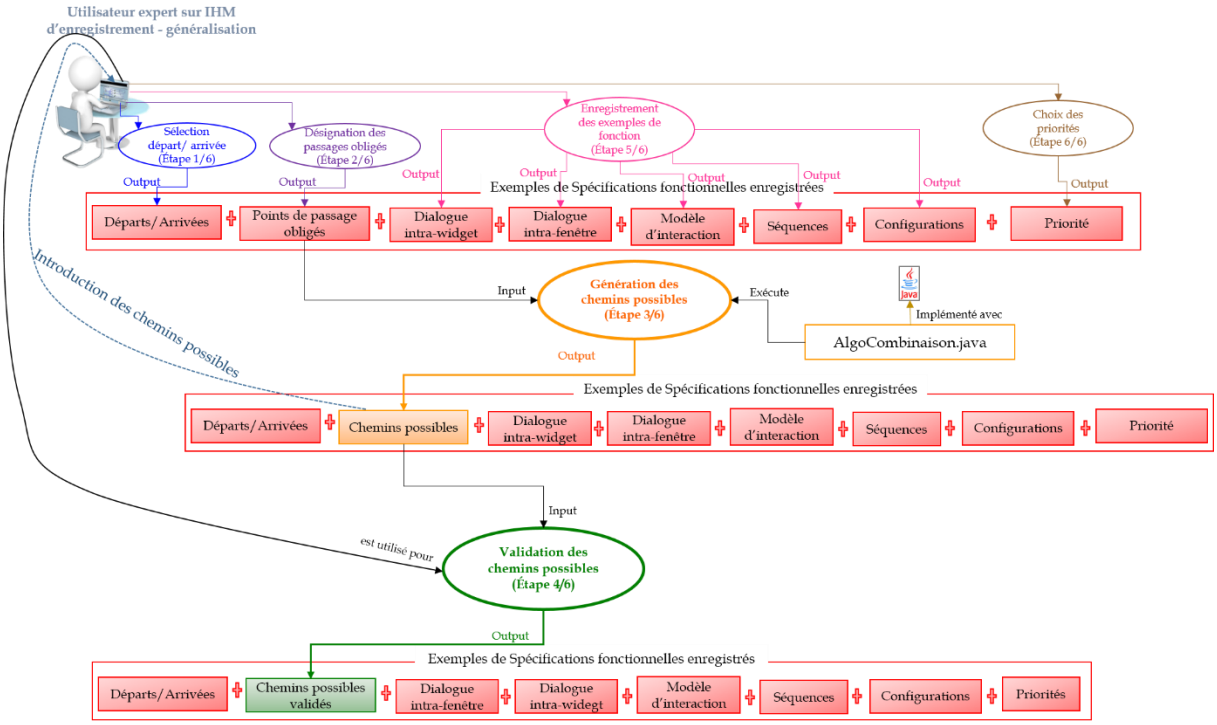


Figure 87 La phase d'enregistrement

- La première étape est la « Sélection départ/Arrivée ». Il s'agit lors de cette étape d'indiquer pour chaque fonction les éléments pouvant lui servir de « départ », puis ceux pouvant être utilisés comme « arrivée ». À la fin de cette étape nous obtenons une liste des « Départs/Arrivées » (sur Figure 87).
- La deuxième étape est la « Désignation des passages obligés ». Un point de passage obligé est un élément incontournable pour l'exécution d'une fonction.
- La troisième étape est la génération des chemins possibles. Les points de passages obligés peuvent être utilisés seuls ou en parallèle. Lors de cette étape, un algorithme de combinaison est mis en œuvre pour proposer tous les points de passage possibles à l'utilisateur expert. Cet algorithme se base sur la combinaison (Équation 1) et sur l'utilisation du fichier contenant tous les points de passages obligés du système, qui sont également enregistrés lors de la description des exemples. Les points de passages obligés sont les éléments nécessaires à la réalisation d'une fonction.
  Ici le but est de faire la combinaison des éléments obligés pour avoir toutes les combinaisons de chemins possibles afin de les proposer à l'expert. Ces combinaisons doivent être validées par ce dernier. Le programme Combinaison produit également un modèle de combinaison qui sera utilisé par le solveur.

$$C_p^n = \frac{n!}{(n-p)!\,p!}$$

Équation 1 Formule de combinaison

  On appelle ***P*** combinaison de ***E tout sous ensemble de E de cardinal p. Cpn ou (np) est*** le cardinal de ***l'ensemble des P combinaisons de E.*** L'objectif est de proposer à l'utilisateur expert tous les chemins possibles.
- Au cours de la quatrième étape, l'ensemble des combinaisons peut alors être sélectionné par l'utilisateur expert ou seulement celles qui lui semblent pertinentes. À la fin de cette étape, on obtient les chemins possibles vérifiés et validés par l'utilisateur expert.
- La cinquième étape est l' « Enregistrement des exemples de fonction ». Il s'agit lors de cette étape d'enregistrer toutes les séquences d'actions nécessaires pour lancer une

fonction, sans utiliser de commande de haut-niveau. Cette étape est composée de trois parties :

  - La première partie de cette étape est la « Configuration du circuit ». Il s'agit lors de cette étape d'enregistrer pas à pas un exemple des actions à faire pour réaliser une fonction en désignant : le ou les points de départ, le ou les points d'arrivée, puis l'enchainement des actions sur le circuit, en respectant la séquence d'action indiquée par les règles de conduite définies pour le système à concevoir. Au cours de cette étape l'utilisateur expert est guidé par les informations issues du modèle de tâches de la fonction de haut-niveau en cours de spécification.
  - La deuxième partie est le « Renseignement de la condition de fin ». C'est à cette étape que l'utilisateur définit la condition pour laquelle la fonction sera arrêtée automatique. Il indique également si la réalisation de la fonction peut être stoppée par l'opérateur avant d'atteindre la condition de fin.
  - La troisième partie est l'« Arrêt des éléments du circuit ». Cette étape permet de retourner le système dans son état initial et d'identifier les séquences d'actions réalisées par l'utilisateur expert pour atteindre cet état.

- La sixième étape correspond au « Choix des priorités ». Il s'agit ici de renseigner la priorité accordée à la fonction par rapport aux autres fonctions du système, pour qu'en cas d'utilisation de ressources communes l'opérateur et/ou le système puisse prioriser la fonction à exécuter.

Lors de l'enregistrement, les étapes trois, quatre et cinq sont réitérées pour permettre l'enregistrement d'un deuxième exemple de réalisation d'une même fonction, allant de la configuration du circuit jusqu'à l'arrêt des éléments utilisés dans cette configuration. En effet, l'enregistrement de deux exemples est nécessaire pour la mise en œuvre de la généralisation.

À l'issue de la phase d'enregistrement, on obtient des exemples de spécifications fonctionnelles qui contiennent pour la fonction spécifiée, les Départ/Arrivée, les chemins possibles, les dialogues intra-widget, les dialogues intra-fenêtre, les modèles d'interaction, les séquences et les configurations des éléments manipulés (Figure 87).

6.2.2. La phase de généralisation

Les différents modèles issus de l'enregistrement sont utilisés dans un module générique de généralisation lié à l'IHM d'enregistrement-généralisation. Deux méthodes sont implémentées dans ce module de généralisation :

- La méthode **AlgoScript** permet de généraliser les actions (étape 1 Figure 88) de l'utilisateur expert enregistrées sous forme de programmes lors de la description des exemples. Il s'agit d'une part, de généraliser les programmes enregistrés décrivant les actions (Dialogue intra-widget) faites par l'utilisateur expert en suivant les étapes décrites dans le modèle de tâches de la fonction en cours de spécification. D'autre part, l'ensemble des séquences d'actions issues des étapes trois, quatre et cinq de la phase d'enregistrement est généralisé pour permettre d'avoir un programme générique régissant le fonctionnement du système à concevoir.
- La méthode **AlgoConfiguration** permet de généraliser les configurations (étape 2 Figure 88). La généralisation des configurations suit plusieurs étapes. Tout d'abord, les chemins possibles validés par l'expert et les Départs/Arrivées enregistrés sont utilisés

dans le MAC (une représentation du système sous forme de graphes et d'un ensemble de contraintes) combiné à un solveur pour obtenir la liste des éléments sur chacun des chemins considérés. Le **solveur** est basé sur l'API GLPK [26] (GNU Linear Programming Kit) qui est un outil performant pour résoudre des problèmes d'optimisation linéaire par recherche de chemin dans un graphe. La liste des éléments obtenue est ensuite combinée avec la nomenclature pour obtenir l'ensemble des configurations possibles qui peut être exploité par la fonction spécifiée.

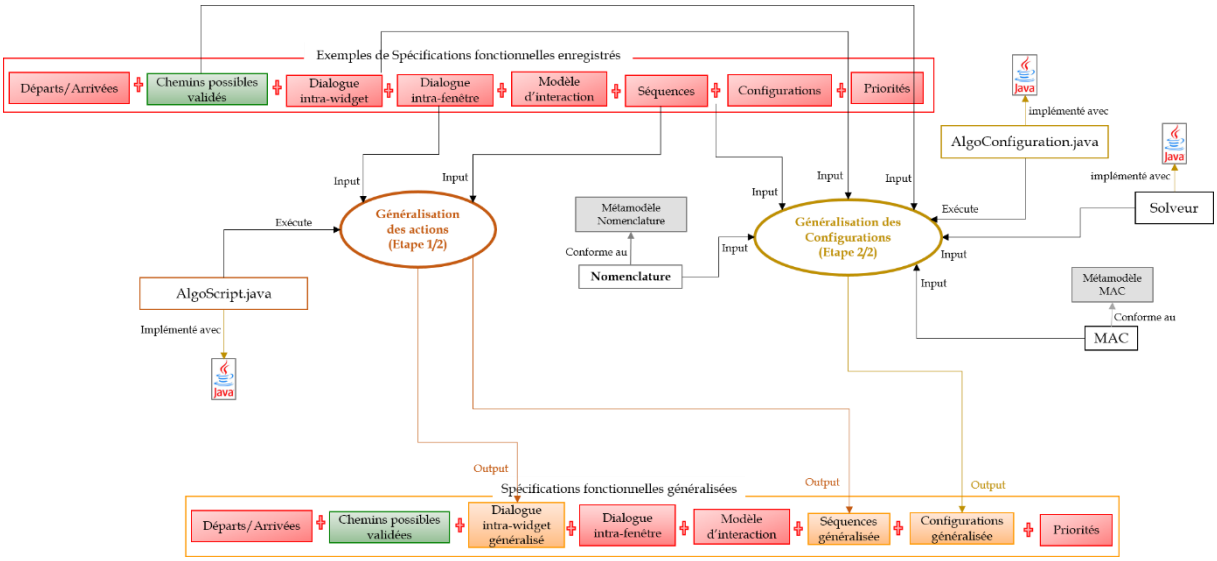


Figure 88 Détail de la démarche de généralisation proposée

À l'issue de ces deux étapes, nous obtenons les spécifications fonctionnelles généralisées (Figure 88) composées de programmes génériques et des configurations généralisées.

7. Opération de génération d'interfaces de contrôle

Les spécifications fonctionnelles généralisées issues de l'opération précédente contiennent une partie des informations nécessaires à la conception des interfaces de contrôles. Les interfaces de contrôles sont composées d'un ensemble de widgets qui facilitent l'exécution des fonctions de haut-niveau à l'opérateur en supervision. La sixième opération de notre flot s'attache à générer ces interfaces.

L'opération de génération des interfaces de contrôle (Figure 89) reçoit en entrée une bibliothèque de widgets, les modèles de tâches et les spécifications fonctionnelles précédemment obtenus pour produire en sortie les interfaces de contrôle.

La transformation permettant de générer les interfaces de contrôle est détaillée sur la Figure 90. Cette transformation se compose de quatre grandes étapes exécutées successivement.

La première étape (**Génération du dialogue inter-fenêtre** sur Figure 90) vise à identifier à partir des opérateurs temporels utilisés dans les modèles de tâches, les différentes fenêtres qui peuvent être contenues dans une interface de contrôle ainsi que les relations entre ces fenêtres (**Dialogue inter-fenêtre**). Le regroupement de ces fenêtres est connu sous le terme « ensemble de présentation (**PTS**) » décrit au chapitre 3 (section 3.3.1.1 pp. 105).

[26] GLPK : https://www.gnu.org/software/glpk/

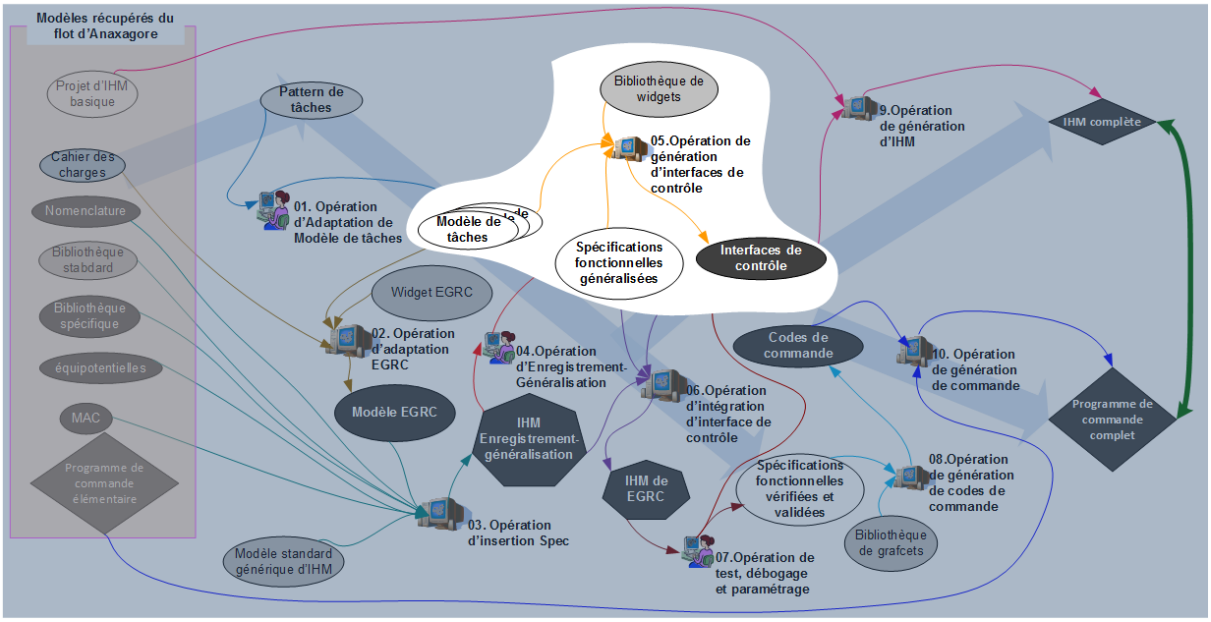


Figure 89 Opération de génération de commande de haut-niveau

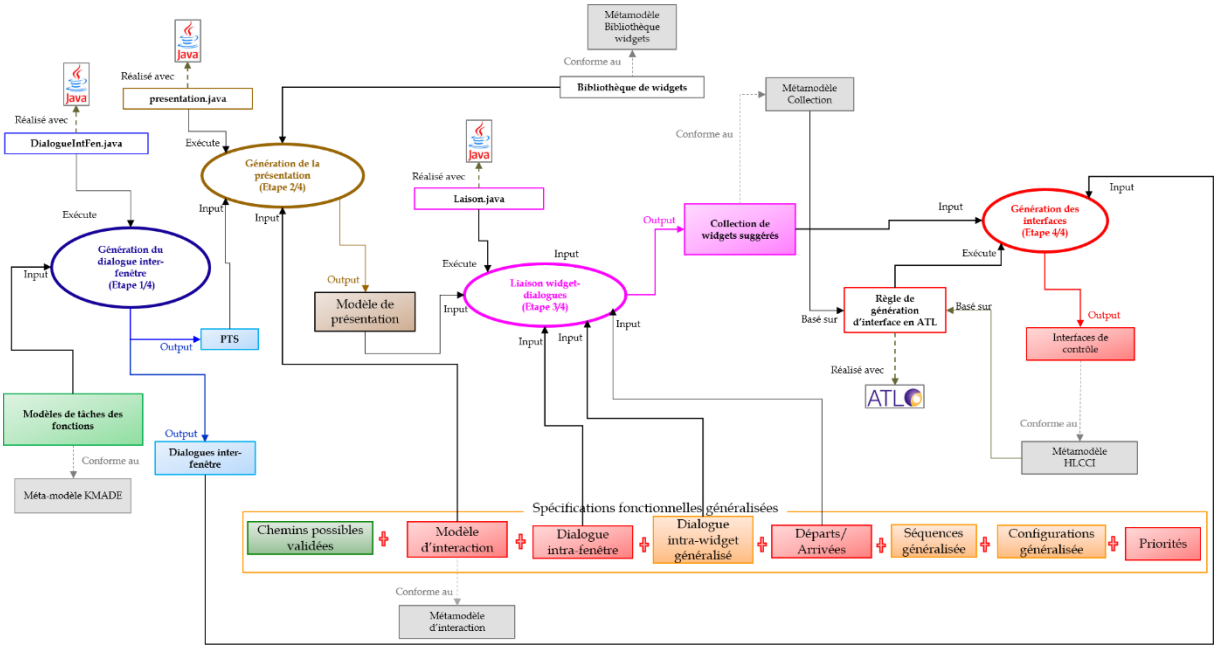


Figure 90 Détail de la génération des interfaces de contrôle pour les contrôle-commandes de haut-niveau

Deux méthodes sont implémentées pour permettre la génération du dialogue inter-fenêtre :

- La méthode **PTSIdentification** analyse l'arbre de tâches à partir de la tâche principale pour rassembler les tâches systèmes élémentaires et les tâches interactives élémentaires en fonction de la décomposition de leur tâche mère (alternative, parallèle, séquentielle, élémentaire, etc.) et du niveau hiérarchique de la tâche fille (numéro de la tâche). Un ensemble de présentations est créé à chaque fois qu'une tâche fille dont la mère est séquentielle est rencontrée. Les autres tâches se trouvant au même niveau dont la mère n'a pas une décomposition séquentielle sont introduites dans le PTS créé.
- La méthode **HeuristiqueApplication** applique les heuristiques identifiées dans les travaux de (Samaan 2006) pour affiner le nombre et le contenu des PTS issus de la première méthode. Ces heuristiques sont présentées dans le Chapitre 3 (section 3.3.1.1 pp. 105).

La deuxième étape (**Génération de la présentation** sur Figure 90) vise à exploiter les ensembles de présentation issus de la première étape pour obtenir un modèle de présentation. Dans ce modèle, les noms des tâches contenues dans les PTS précédemment obtenus sont associés à des widgets graphiques. L'implémentation de cette deuxième étape a conduit à la définition de deux méthodes :

- La méthode **IdentifyWidget** identifie dans le modèle d'interaction pour chaque nom de tâche contenu dans chaque PTS, le type de widget associé. On obtient ainsi un modèle intermédiaire contenant des ensembles dont chaque élément est identifié par le couple (nom de tâche, type de widget).
- La méthode **GetWidget** récupère de la bibliothèque de widgets, la représentation graphique correspondant aux types de widget identifiés dans le modèle intermédiaire précédemment obtenu.

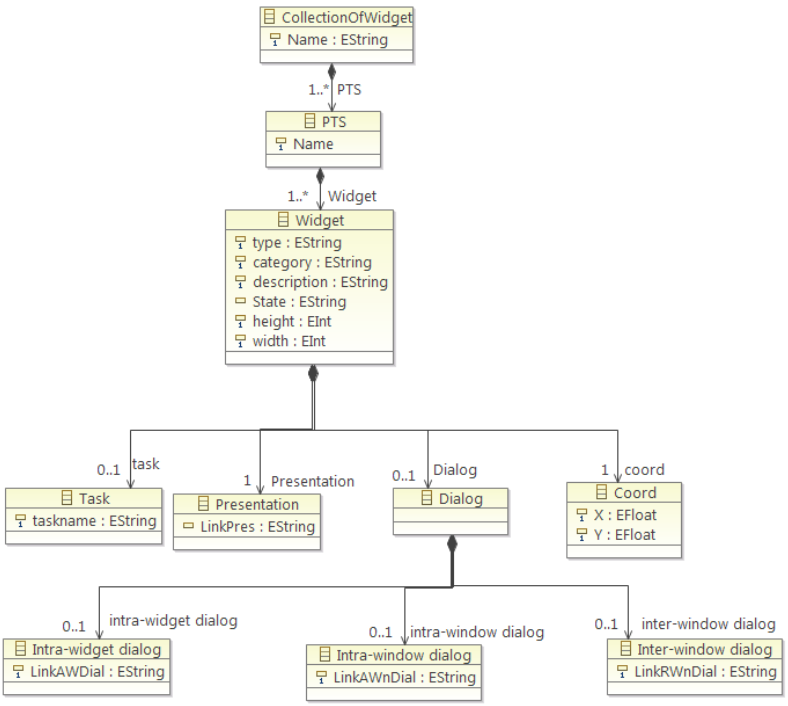


Figure 91 Méta-modèle Collection de Widgets

À l'issue des deux étapes précédentes, on obtient un regroupement de la partie statique des widgets dans les fenêtres identifiées. L'étape suivante, la troisième (*Liaison widget-dialogue sur* Figure 90), consiste à associer dans chaque ensemble de widgets précédemment obtenus, le dialogue correspondant aux widgets. Il s'agit ici d'associer les Dialogues intra-fenêtre, les Dialogues intra-widget généralisés ainsi que les Départ/arrivée aux widgets des ensembles de présentation, pour obtenir une **collection de widgets suggérés** (Figure 90). Cette collection donne une idée de l'organisation dynamique des widgets d'une même fenêtre et est générée conformément au méta-modèle de la Figure 91. Deux méthodes sont implémentées :

- La méthode **AssociateDialog** parcourt le modèle de présentation précédemment obtenu et identifie à partir du nom de tâche, le lien vers les dialogues enregistrés et ou généralisés.
- La méthode CreateCoord sert à calculer les dimensions des widgets d'une même fenêtre et à déterminer leurs positions en utilisant les dialogues intra-fenêtre.

La dernière étape (**Génération des interfaces** sur Figure 90) génère les interfaces de contrôle. Une transformation ATL est implémentée pour créer la structure visuelle des interfaces de contrôle telle que perçue par l'opérateur en supervision. Cette structure contient l'organisation dynamique des fenêtres identifiées pour une même fonction de haut-niveau et aussi pour l'ensemble des fonctions de haut-niveau du système à concevoir.

La transformation implémentée pour cette étape se compose de trois règles :

- La règle **Collection2Interface** traite les informations relatives au nom des fonctions enregistrées. Elle permet également la création d'un widget de type Display et d'un widget de type Conteneur. Le widget de type Display permet d'afficher un historique sur les fonctions de haut-niveau réalisées par l'opérateur en supervision.
- **PTS2CommandWidget** permet d'identifier dans les PTS précédemment obtenus et en fonction des noms de tâche associés, les widgets correspondant aux *CommandWidget*. Elle est exécutée pour chaque fonction de haut-niveau spécifiée.
- **PTS2ControlInterface** permet d'identifier dans les PTS précédemment obtenus et en fonction des noms de tâche associés, les widgets correspondant aux *ControlInterface*. Elle est exécutée pour chaque fonction de haut-niveau spécifiée.

8. Opération d'intégration des interfaces de contrôle

Les interfaces de contrôle précédemment obtenues contiennent des informations issues des spécifications fonctionnelles généralisées qui doivent être vérifiées, corrigées puis validées. Afin de permettre à l'utilisateur expert de vérifier puis de valider les spécifications fonctionnelles, nous utilisons les commandes de haut-niveau dans la septième opération de notre flot (Figure 92) pour générer une IHM d'Enregistreur-Généralisateur-Rejoueur-Correcteur (EGRC).

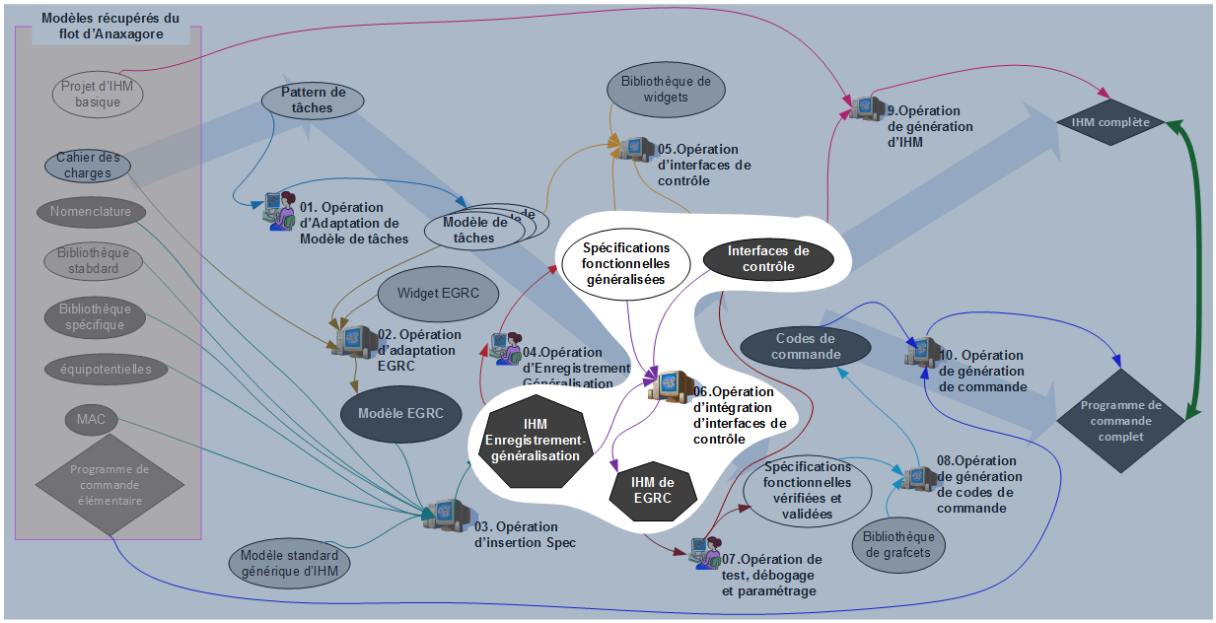


Figure 92 Opération de génération d'IHM d'EGR

L'opération d'intégration des commandes de haut-niveau est détaillée sur la Figure 93. Il s'agit d'intégrer dans le rejoueur disponible sur l'IHM d'enregistrement-généralisation les interfaces de contrôle générées ainsi que les séquences d'actions et configurations généralisées qui permettent de tenir compte des retours du système aux actions de l'utilisateur.

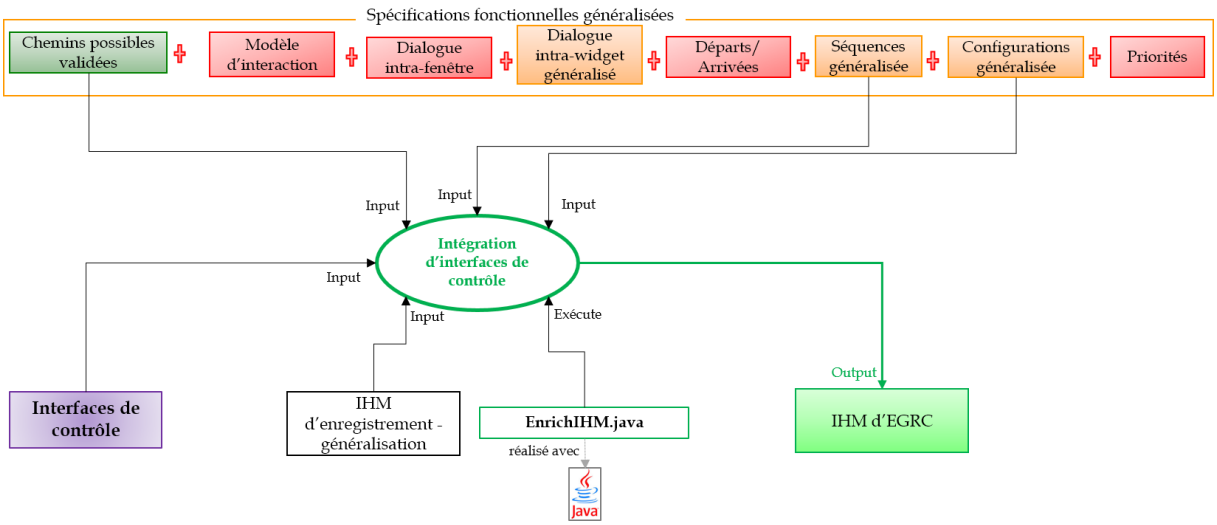


Figure 93 Détail de la génération de l'IHM d'EGRC

9. Opération de test, débogage et correction

L'opération de **test, débogage et correction** (Figure 94) vise à obtenir d'une part des spécifications fonctionnelles vérifiées et validées (Figure 96). D'autre part, cette opération exploite l'expertise de l'ergonome pour modifier l'apparence des interfaces de contrôle générées (Figure 97).

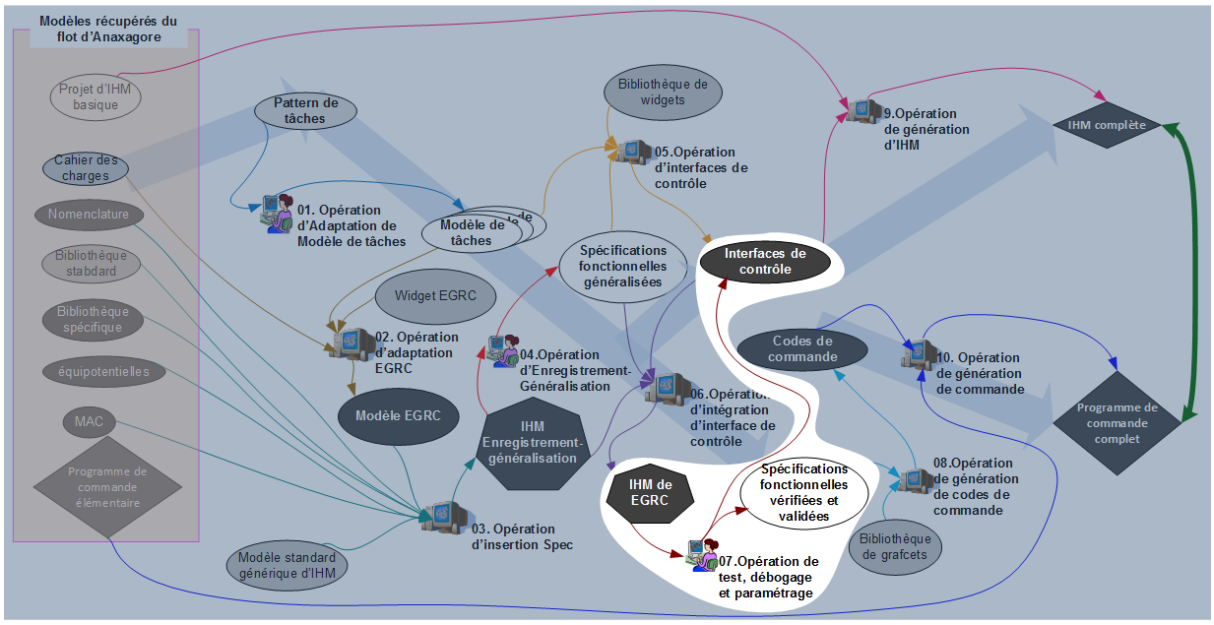


Figure 94 Opération de test, débogage et paramétrage

Lors de la phase de rejeu (vérification et la validation) l'expert choisit une fonction à rejouer dans la liste de fonctions qu'il a spécifiée (Figure 95). Cette liste se met à jour au fur et à mesure que l'expert enregistre les fonctions du système. Seules les fonctions enregistrées sont contenues dans cette liste. Une fois la fonction sélectionnée, le système affiche l'interface de contrôle générée correspondant à cette fonction. Cette interface permet à l'expert de rejouer toutes les configurations généralisées. Lorsqu'une configuration est juste, l'expert peut la valider pour permettre plus tard la prise en compte seulement des configurations qu'il a validées. Par contre lorsqu'une configuration est fausse, il peut la modifier. La configuration à modifier s'affiche sur le synoptique du système pour permettre à l'expert de la modifier en

manipulant les éléments du système comme lors de l'enregistrement de la fonction. Seules les configurations validées par l'utilisateur expert sont enregistrées et utilisées dans la conception du système de contrôle-commande.

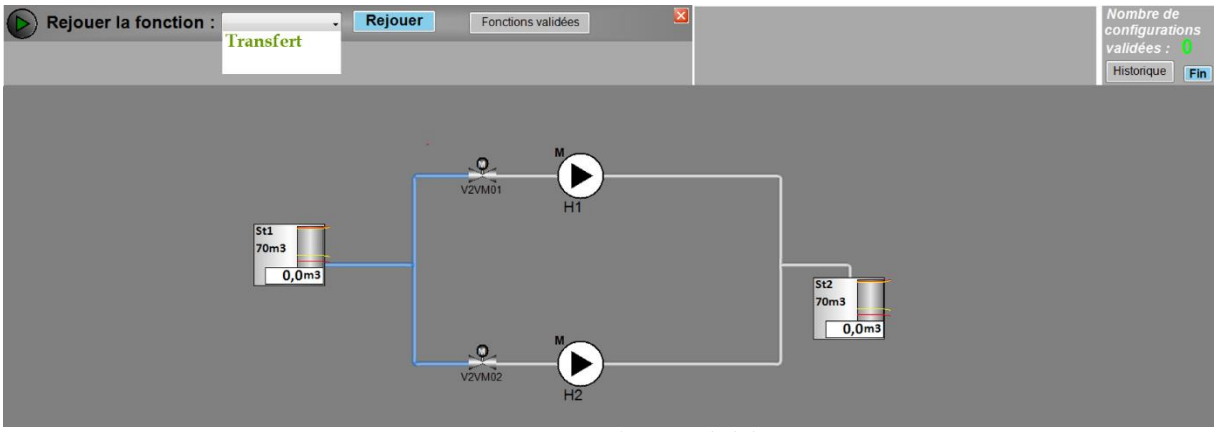


Figure 95 IHM de rejeu générée

La Figure 95 montre l'IHM d'EGRC lorsqu'on clique sur le rejoueur. On y retrouve le synoptique du système de transfert d'eau douce, ainsi que la liste des fonctions spécifiées qui ne contient que Transfert, dans notre cas. Une fois la fonction choisie dans la liste, l'utilisateur expert peut cliquer sur le bouton « Rejouer » et suivre les instructions de l'interface pour rejouer les configurations et programmes généralisés, afin de vérifier et de valider les configurations.

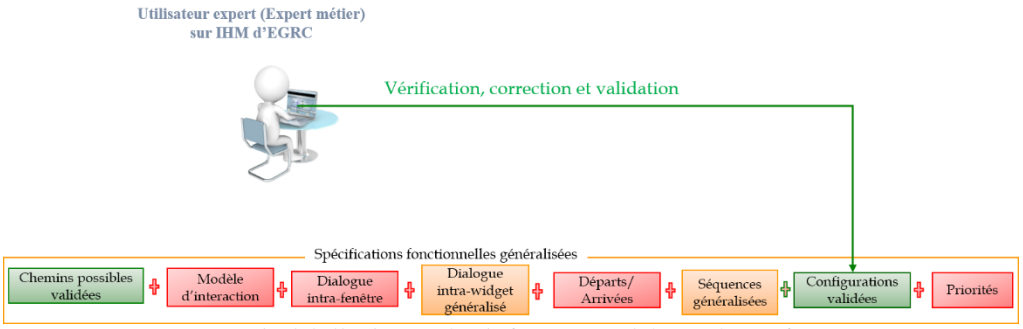


Figure 96 Détail de l'opération de vérification et validation des configurations

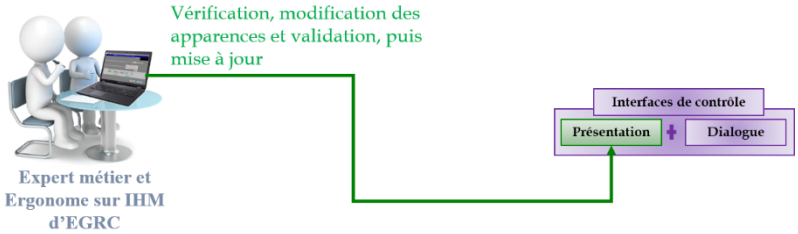


Figure 97 Vérification et validation de la présentation des interfaces de contrôle

En complément du rejeu et de la correction des configurations, le système EGRC propose une interface de modification de la présentation des interfaces de contrôle générées. Cette interface est utilisée par l'utilisateur expert assisté de l'ergonome pour modifier, s'ils ne sont pas adaptés à la tâche à réaliser, les widgets suggérés lors de l'enregistrement des exemples. L'objectif étant de rendre plus ergonomiques les présentations générées pour chaque interface de contrôle (Figure 97). À chaque modification de présentation, les interfaces de contrôle sont mises à jour.

10. Opération de génération de codes de commandes

Les configurations validées et séquences généralisées sont utilisées dans la neuvième opération de notre flot pour générer les codes de commande (Figure 98).

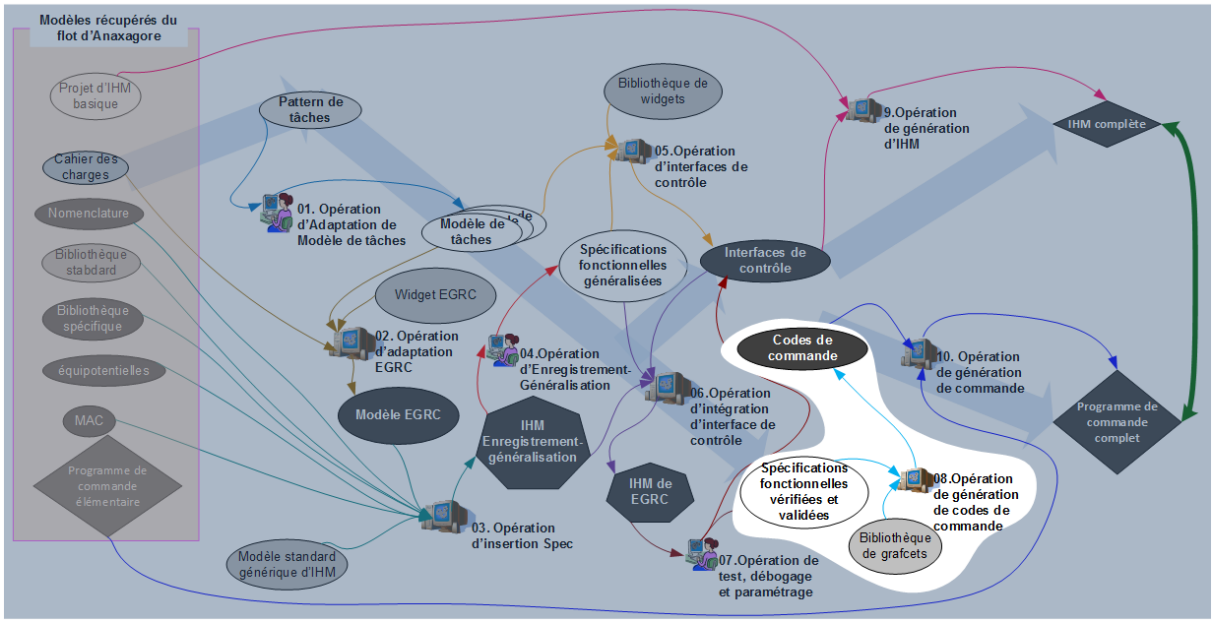


Figure 98 Opération de génération de codes de commande

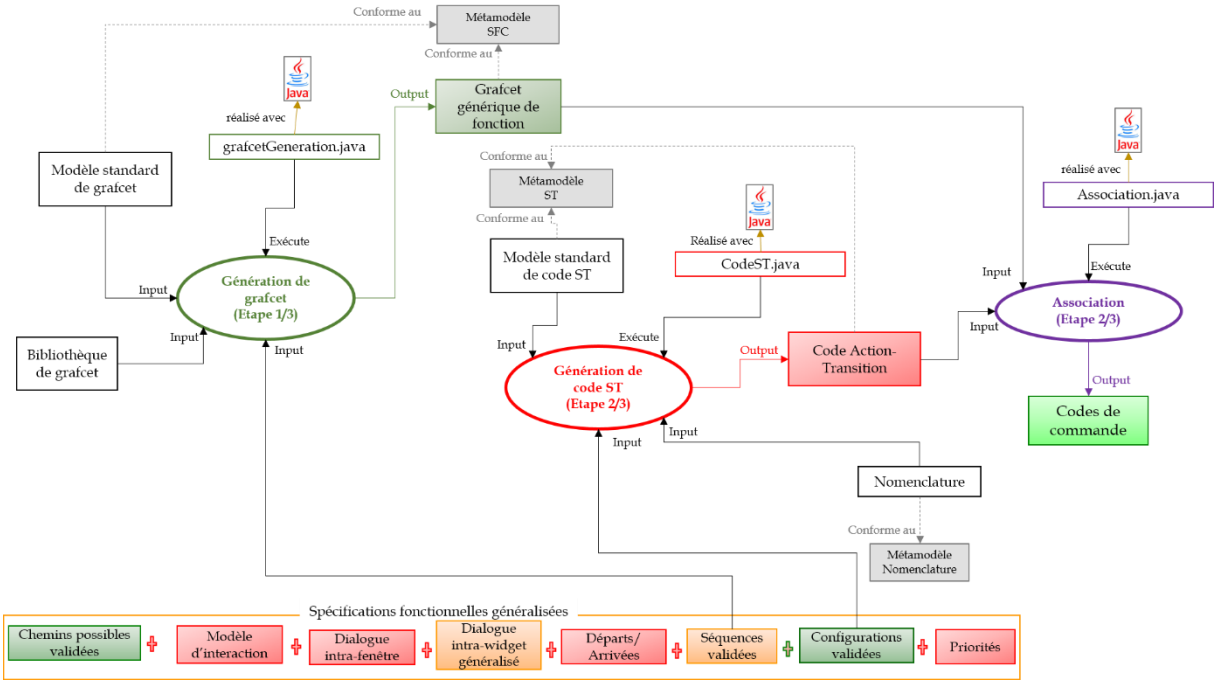


Figure 99 Détail de la génération de codes de commande

L'opération de génération de codes de commande est détaillée (Figure 99). Elle reçoit en entrée les configurations et séquences vérifiées puis validées par l'utilisateur expert et une bibliothèque de grafcet pour produire en sortie les codes de commande. Cette transformation se compose de trois grandes étapes exécutées successivement.

La première étape de cette démarche (***Génération de Grafcet*** sur Figure 99) vise à obtenir le Grafcet générique qui décrit les étapes et les liaisons entre ces étapes. Elle reçoit en entrée un

modèle standard de Grafcet, les séquences d'actions élémentaires généralisées et une bibliothèque de SFC pour produire en sortie un grafcet générique de fonction.

Le *modèle de Grafcet* est un fichier SFC vide créé sous Straton. Ce modèle est le point de départ de notre processus de génération. Les éléments de la *bibliothèque de Grafcet* sont instanciés dans ce modèle conformément aux séquences généralisées. La *bibliothèque de Grafcet* contient les objets permettant de construire un Grafcet : les étapes, les transitions, les liaisons orientées. Les *séquences d'actions généralisées* contiennent les actions élémentaires nécessaires pour le lancement d'une commande globale. Elles permettent de connaître l'enchainement des actions élémentaires pour la réalisation de la commande globale. Ces informations sont déduites des suites d'actions effectuées par l'expert pour la réalisation des fonctions lors de l'opération de spécification fonctionnelle.

La génération du grafcet générique de fonction est effectuée en respectant la règle d'établissement du grafcet qui stipule que chaque liaison orientée relie une étape à une transition ou une transition à une étape. Un grafcet se lit de haut en bas. Le non-respect de cette règle entraine une erreur dans l'application.

La deuxième étape (***Génération de code ST*** sur Figure 99) vise à obtenir les codes Action-Transition qui doivent être associés aux Grafcets. Cette étape reçoit en entrée le modèle standard de code ST, les configurations et alertes généralisées issues de l'opération de spécification fonctionnelle (voir section 3.2 au Chapitre 3) et la nomenclature. Elle produit en sortie le modèle de code Action-Transition en langage ST. Le modèle standard de code ST est un fichier de programmes ST vide créé sous Straton. Les configurations généralisées contiennent l'état des variables de contrôle issues de la supervision liées aux éléments du système, nécessaires à la réalisation d'une fonction. Ces informations sont insuffisantes pour créer les lois de commande. Il est nécessaire de connaître toutes les variables échangées entre la supervision et la commande. De la nomenclature, on récupère toutes les variables échangées entre la supervision et la commande qui correspondent aux configurations généralisées. L'ensemble des configurations et des variables récupérées de la nomenclature vont permettre de décrire en ST les actions et les répétitivités liées aux transitions. Cette étape est implémentée à travers les trois méthodes suivantes :

- La méthode **CreateStep** sert à créer dans le modèle Standard de ST, le code ST qui doit être associé aux étapes du grafcet à partir des configurations validées.
- La méthode **VariablesTable** sert à créer à partir des configurations validées et de la nomenclature, une table de variables nécessaire à la gestion des éléments de la chaine de contrôle-commande.
- La méthode **CreateTransition** sert à créer les différentes transitions relatives aux informations contenues dans la table précédemment créée.

Les informations obtenues dans l'étape précédente seront associées par appel des méthodes du *code Action-Transition*, aux actions et aux transitions décrites dans le Grafcet générique de fonction : c'est la troisième étape (***Association*** sur Figure 99). En effet, pour être utilisé avec l'interface de contrôle-commande global, le grafcet générique de fonction est adapté aux différentes actions-transitions de chaque fonction. Le grafcet générique de fonction permet une représentation des actions séquentielles élémentaires intervenant dans la réalisation d'une fonction.

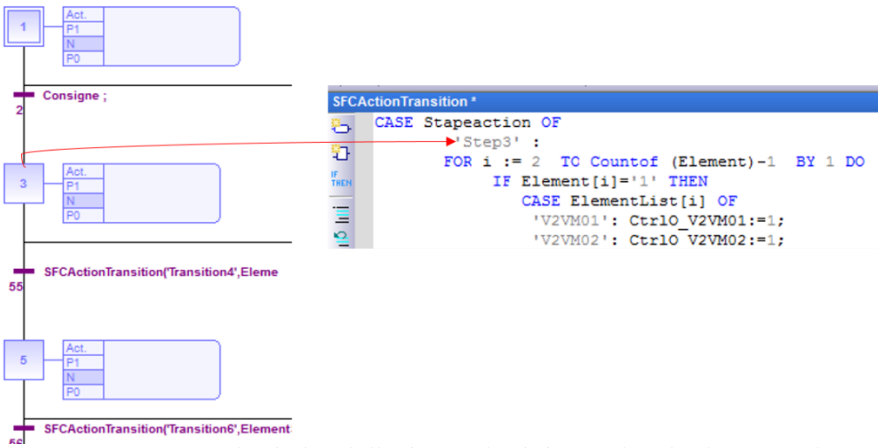


Figure 100 Extrait du résultat de l'opération de génération de codes de commande

La Figure 100 présente un extrait de grafcet générique obtenu après avoir déroulé les trois étapes de l'opération de génération de codes de commande.

11. Opération de génération d'IHM

La dixième opération de notre flot réalise l'insertion des interfaces de contrôle contenues dans les contrôle-commandes globaux dans le projet d'IHM basique. Aucune commande de haut-niveau n'est pour l'instant prise en charge sur le projet d'IHM basique. L'opération de génération d'IHM reçoit en entrée le projet d'IHM basique et les interfaces de contrôle, et produit en sortie l'IHM globale (Figure 101). Nous définissons les termes IHM basique et IHM complète avant de détailler la transformation utilisée dans cette opération.

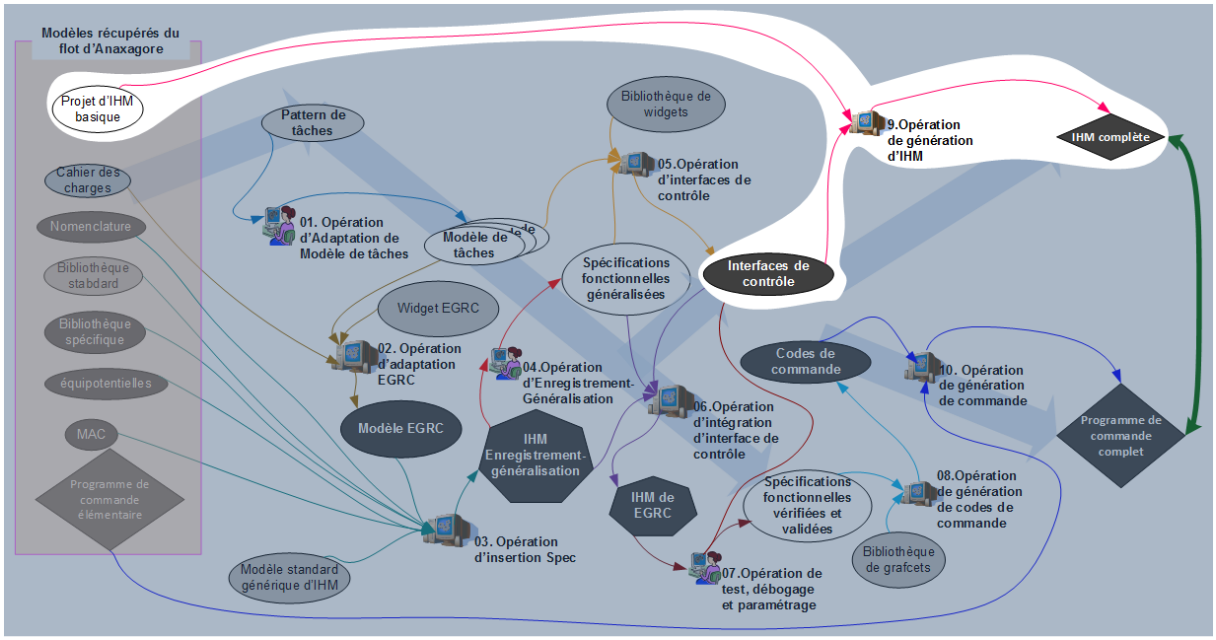


Figure 101 Opération de génération d'IHM

11.1. Projet d'IHM basique

Le **projet d'IHM basique** (Figure 102) est l'application de supervision générée à partir des travaux existant (Bignon 2012). Il offre une interface graphique représentant le synoptique du système à concevoir. Cette interface permet une manipulation de bas niveau de tous les éléments du système (par exemple le clic sur un élément, l'ouverture ou la fermeture d'un élément). Elle sera utilisée dans la génération de l'IHM globale du système.

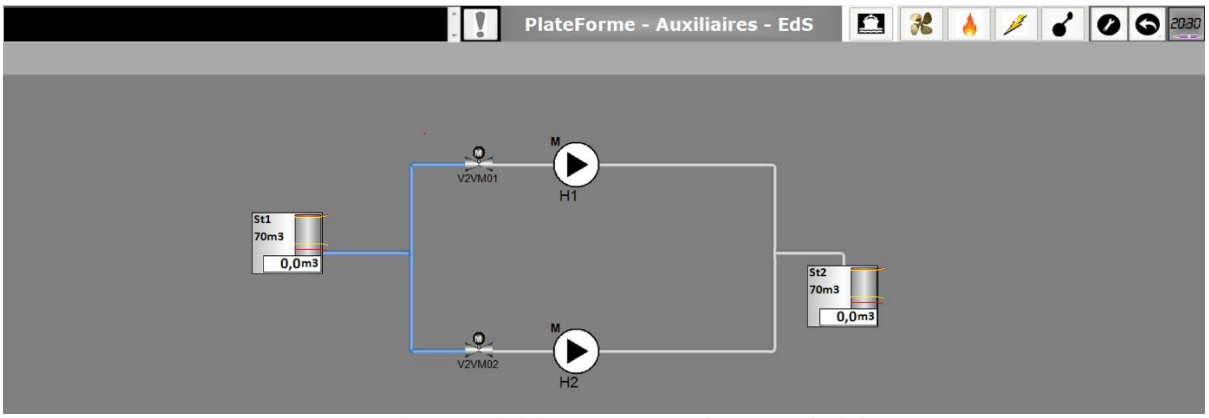


Figure 102 IHM basique générée pour notre système exemple de la Figure 63

Comme sur la zone de visualisation de la Figure 85, la Figure 102 présente également dans sa zone de visualisation un système possédant les dispositifs de stockage (soutes St1 et St2) et de distribution (groupes hydrophores H1 et H2), conformément au schéma P&ID de la Figure 63. L'IHM basique permet notamment de jouer des scénarios de transfert d'eau de la soute St1 et à la soute St2. Dans le logiciel Panorama E2, elle se compose d'un nombre important d'objets de base du logiciel, que l'on peut assimiler à des widgets généralement connectés à un programme de contrôle-commande.

Cette interface contient en haut à gauche une zone d'alarme en (noir) et des widgets en haut à gauche décrits dans (Rechard 2015), permettant de naviguer entre les sous-systèmes d'un navire.

11.2. L'IHM complète

L'**IHM complète** (Figure 103) est un modèle opérationnel généré automatiquement. C'est l'IHM finale qui sera utilisée à bord du navire. En plus de l'interface graphique qui reprend la représentation du synoptique de l'IHM basique, cette IHM contient également les interfaces des contrôle-commandes de haut-niveau qui permettront à l'opérateur, d'atteindre plus facilement les objectifs de haut niveau du système.

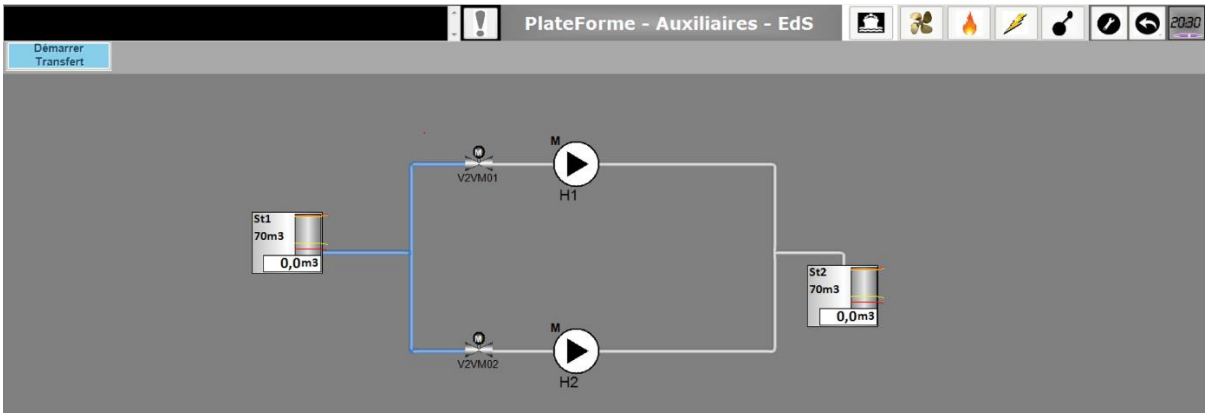


Figure 103 Résultat de l'opération de génération d'IHM

Sur la Figure 103, l'IHML complète reprend la structure de l'IHM basique. On y retrouve également le bouton permettant de lancer la fonction de *Transfert*. Ce bouton contient la commande de haut-niveau précédemment générée. L'IHM complète permet toujours une surveillance des éléments du système identifiés sur le synoptique. De plus, le contrôle de manière globale par déclenchement de séquences regroupant un ensemble de commandes élémentaires est possible, à travers le bouton « Transfert ».

11.3. Transformation de génération d'IHM

La génération automatique de l'IHM complète est réalisée suivant deux grandes étapes. Celles-ci ont pour but d'enrichir l'architecture et la structure du projet d'IHM basique (Figure 104).

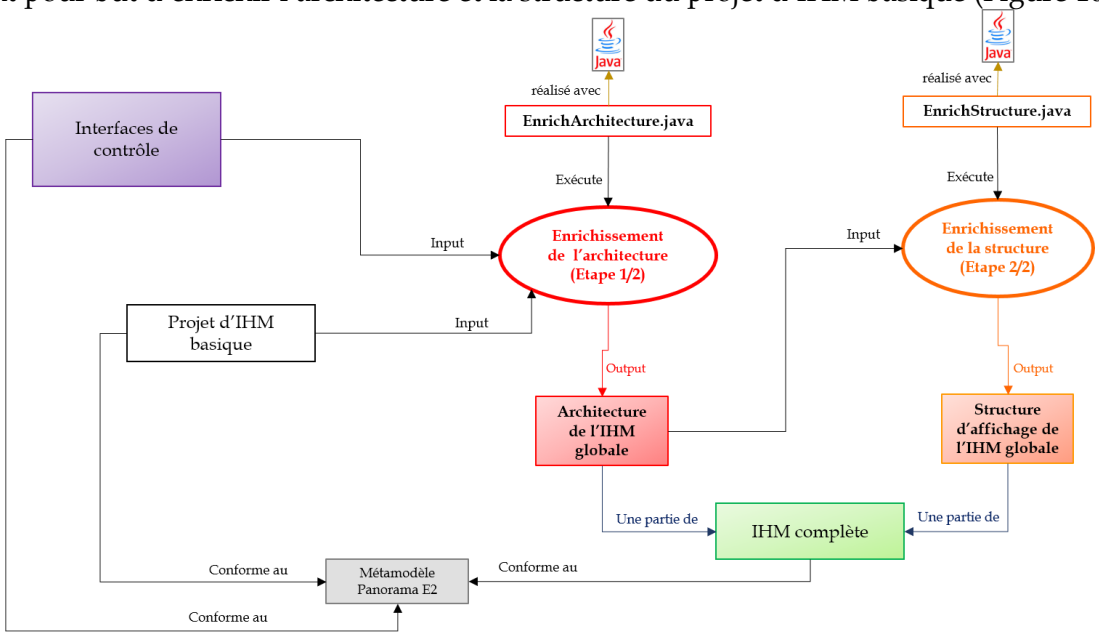


Figure 104 Détail de l'opération de génération d'IHM

L'insertion des interfaces de contrôle est réalisée conformément à la structure du logiciel de supervision que nous avons choisie (Voir section 2.1). Quatre méthodes sont définies :

- La méthode **EnrichissementArchitecture** permet d'enrichir le fichier Unit.cfg décrivant l'architecture du système selon les différentes fonctions spécifiées par l'expert.
- La méthode **EnrichissementStructure** permet d'enrichir le fichier Canevas.cfg décrivant la structure du système selon les différentes fonctions spécifiées par l'expert.
- La méthode **AjoutInterfaceControle** permet de créer un dossier « Fonction » et d'y copier les sous dossiers portant le nom des fonctions et contenant les interfaces de contrôle générées, vérifiées, puis validées.
- La méthode **Enregistrer** permet, à chaque modification, d'enregistrer ou de créer le fichier considéré.

12. Opération de génération de commande

La onzième opération de notre flot réalise l'insertion des codes de commande contenus dans les commandes de haut-niveau dans un programme de commande élémentaire. Aucune commande de haut niveau n'est à ce stade prise en charge dans le programme de commande élémentaire. L'opération de génération de commande reçoit en entrée le programme de commande élémentaire et les codes de commande contenus dans commandes de haut-niveau, et produit en sortie le programme de commande complet (Figure 105). Nous définissons les termes programme de commande élémentaire et programme de commande complet avant de détailler la transformation utilisée dans cette opération.

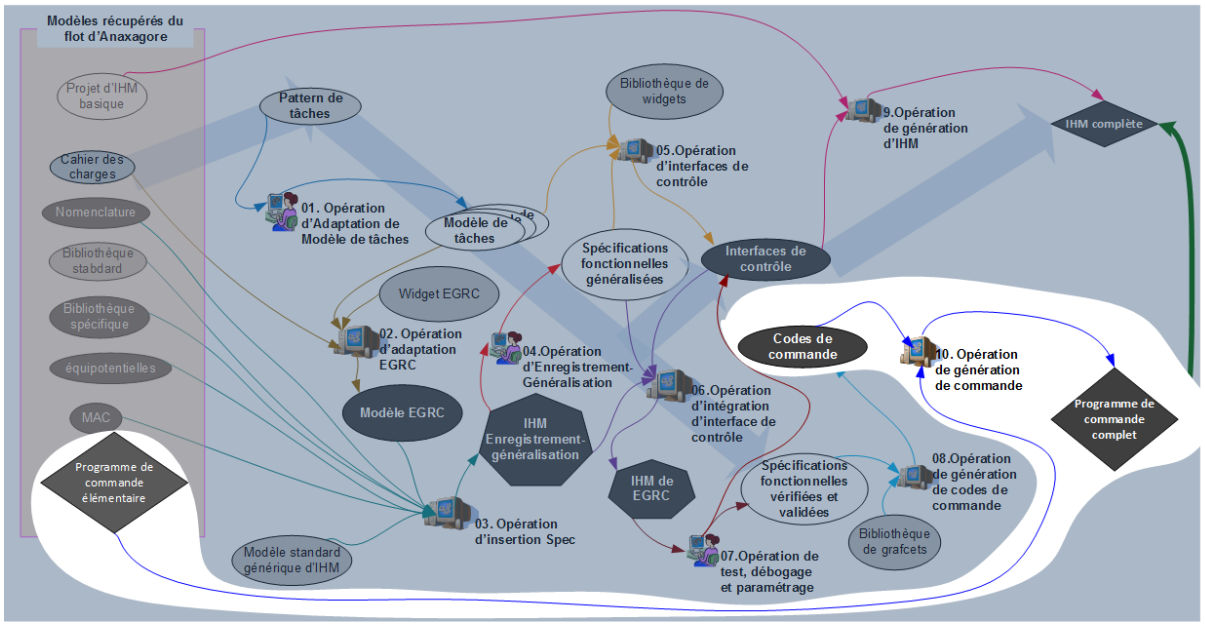


Figure 105 Opération de génération de codes de commande

12.1. Le programme de commande élémentaire

Le programme de commande élémentaire est généré à partir des travaux de (Bignon 2012) et permet le contrôle élément par élément des composants du système. L'obtention de ce programme se fait par assemblage successif des vues de commande des éléments. Ce programme interprétable par Straton est implanté sur des automates et répond aux actions effectuées sur les éléments par l'opérateur à travers l'IHM de supervision.

Le programme élémentaire de commande, généré via *Anaxagore* lors de l'opération d'assemblage (Bignon 2012), permet d'obtenir un programme de commande associé au synoptique établi par l'expert. Celui-ci comporte l'ensemble des vues commandes de chaque élément du système ainsi que leurs variables associées. Il permet ainsi, via le langage FBD (Functional Block Diagram), la commande manuelle des différents éléments du système (ouverture/fermeture, marche/arrêt). Cependant, aucune commande globale propre au système ne peut être générée. Cela fait l'objet de cette onzième opération.

Dans le cas présent, le fichier XML du programme élémentaire, qui reprend la structure du modèle de programme de commande, est utilisé afin d'être enrichi des commandes de haut niveau (ou commandes globales). Nous cherchons donc à ce stade à générer un programme de commandes comportant à la fois des commandes élémentaires propres aux éléments du système mais également des commandes globales propres au système lui-même.

12.2. Le programme de commande complet

Le **programme de commande complet** est un modèle opérationnel généré automatiquement. Ce programme de commande est également interprétable sous Straton, contenant les Grafcets génériques des fonctions permettant l'exécution des fonctions depuis la supervision et le programme de commande élémentaire. Il doit permettre non seulement le contrôle des commandes pour l'exécution des fonctions du système mais aussi le contrôle individuel des éléments du système.

12.3. Transformation de génération de codes de commande

La génération automatique du programme complet est réalisée suivant une seule phase (Figure 106). Trois méthodes sont implémentées :

- La méthode **CreationFonction** permet la création d'un dossier « Fonctions » au sein de l'arborescence du programme de commande élémentaire.
- La méthode **AjoutCodeCommande** permet l'ajout des codes de commande associés à chacune des fonctions spécifiées par l'expert dans le dossier précédemment créé au sein du programme élémentaire.

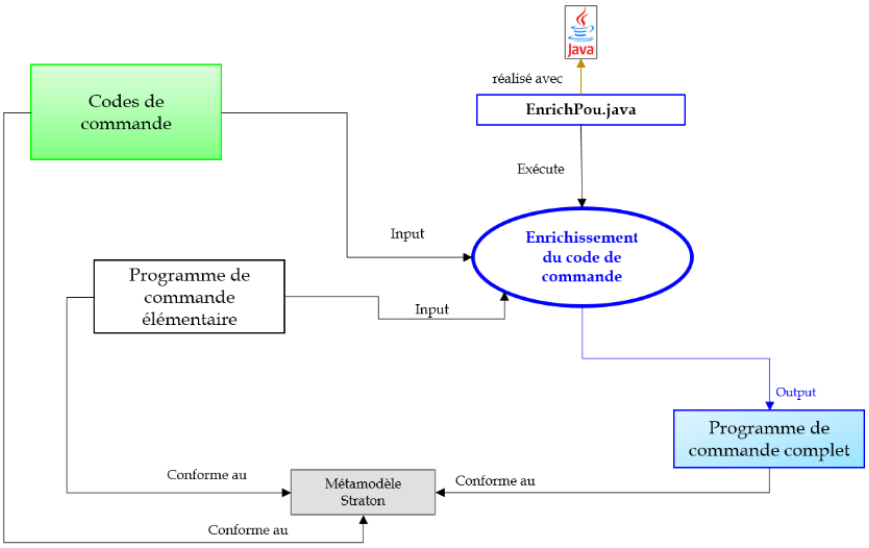


Figure 106 Détail de l'opération de génération de code de commande

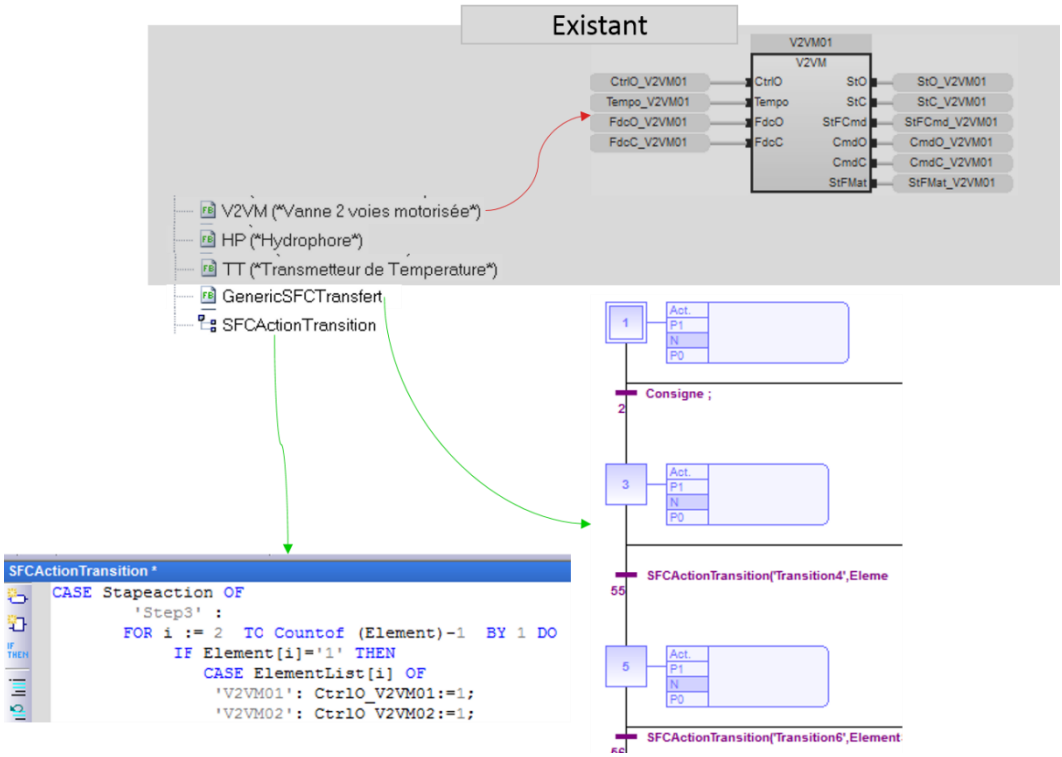


Figure 107 Extrait du résultat de l'opération de génération de commande

À l'issue de cette transformation, nous obtenons un programme de commande complet (Figure 107) contenant à la fois le programme de commande élémentaire et les codes de

commande de haut-niveau permettant de répondre aux consignes envoyées par les opérateurs en supervision pour le lancement d'une commande de haut-niveau.

13. Bilan

Ce chapitre a présenté l'implémentation de notre flot de conception qui concrétise nos propositions dans le processus de spécification de tâches humaines et de spécification fonctionnelle d'un système complexe.

Cette implémentation a permis de proposer des outils concrets à l'utilisateur expert lui permettant de capturer au mieux ses connaissances tant pour la conception des modèles de tâches que pour l'obtention des spécifications fonctionnelles.

La combinaison des techniques de l'EUD et de l'IDM a pour but de réduire l'effort de conception, de spécification et d'intégration des spécifications dans la conception du système de contrôle-commande complet.

L'utilisation des techniques de l'IDM pour générer les différents modèles opérationnels recentre les experts sur leur cœur de métier, sans nécessiter des phases coûteuses d'explications pour d'autres acteurs de la conception. La génération conjointe des codes de commandes et des interfaces de contrôle, doit permettre d'affermir la cohérence entre le programme de commande complet et l'IHM complète, de générer un gain de temps considérable et d'éviter d'éventuelles erreurs. Cette génération se réalise au travers d'une succession d'opérations présentées dans notre flot de conception (Figure 61), qui complètent le flot de conception intégré d'*Anaxagore*.

Toutes les opérations de notre flot de conception sont intégrées à l'outil Anaxagore (section 1.3.3.2 du chapitre 1), développé à des fins de validation de la démarche complète proposée de génération de système de contrôle-commande. Nous présentons dans le prochain chapitre l'application de notre flot de conception à un cas d'application représentatif d'un système industriel ainsi que l'évaluation des interfaces générées afin de les rendre utilisables.

Chapitre 5 : Application de notre démarche à un cas d'étude et validation expérimentale

Nous avons proposé une démarche de conception qui permet, à partir de modèles existants Anaxagore et d'autres modèles intermédiaires, de produire conjointement une IHM complète de supervision et un programme complet. Les différents modèles intermédiaires à la mise en œuvre de notre démarche (modèles de tâches et spécifications fonctionnelles) sont obtenus en capturant la connaissance des experts métiers (utilisateurs experts) sur le système à concevoir.

Pour valider notre démarche, nous avons procédé à deux types d'expérimentation. Dans la première section de ce chapitre, nous décrivons l'application de la méthode à un cas d'expérimentation sur un exemple concret de processus industriel. Puis, afin de vérifier nos choix de conception et de valider nos propositions, les outils de spécification ont été soumis à des évaluations. Deux expérimentations ont été réalisées auprès d'utilisateurs expérimentés par moyen de tests utilisateurs, qui sont présentés dans la *deuxième section* de ce chapitre.

1 Étude de cas

Anaxagore est composé d'un ensemble d'opérations (voir section 1.3.4 au chapitre 1), implémentées sous forme de plugins à la plateforme Eclipse, et de bibliothèques d'éléments. Anaxagore est appliqué à la conception de systèmes de contrôle-commande. Il s'appuie sur plusieurs outils industriels spécifiques communément utilisés dans la conception des systèmes de contrôle-commande. La Figure 108 schématise les liens entre Anaxagore et ces logiciels. *Anaxagore* reçoit en entrée des modèles issus de Microsoft® Visio sous la forme de dessins XML et produit en sortie un programme complet de commande pour Straton et une IHM complète de supervision pour Panorama E2.

Figure 108 Ecosystème de l'outil Anaxagore (Bignon, 2012)

Les différentes opérations de notre flot de conception renvoient des modèles interprétables par ces deux outils. Ces opérations, comme celles qui sont dans *Anaxagore*, sont implémentées sous forme de plugins à la plateforme Eclipse.

1.1 Un exemple concret : le système EdS (Eau douce Sanitaire)

Nous avons appliqué l'ensemble de notre démarche à la conception des systèmes de contrôle-commande dans le domaine maritime. Plus précisément, nous considérons le système de distribution, stockage et production d'eau douce sur un navire (nommé EdS sur Figure 109). Ce type de système alimente le navire en eau douce et en eau déminéralisée pour les moteurs et présente la particularité de posséder une sémantique particulière liée à l'écoulement de

fluides. Le schéma P&ID (Figure 109), partagé par les experts en charge de la conception, est vérifié grâce des techniques de vérification formelle (Mesli-kesraoui, Toguyeni, et al. 2016). Il peut donc être considéré comme une spécification structuro-fonctionnelle formelle de bas-niveau du système.

Ce schéma représente trois soutes de stockage (St1, St2 et St3). Chacune de ces soutes est équipée d'une jauge de niveau, d'un transmetteur de niveau et d'un détecteur de niveau. Ces deux derniers instruments renvoient leurs informations à la supervision.

Ce schéma représente également deux moyens de production (OsmAv et OsmAr). Chacun de ces osmoseurs produit de l'eau douce sanitaire et de l'eau déminéralisée. L'eau déminéralisée est stockée dans la soute St3. La vanne V2vM10 permet d'isoler la soute du reste du circuit. L'eau douce sanitaire est stockée dans les soutes St1 et St2 qui disposent chacune d'une vanne d'isolement en amont (respectivement V2VM01 et V2VM02). L'eau douce sanitaire produite est traitée en amont des soutes par un dispositif de chloration (TRCH01).Chaque dispositif de désalinisation est également connecté à deux autres vannes (V2VM11 et V2VM12 pour OsmAv, V2VM13 et V2VM14) permettant l'aspiration de l'eau de mer et le refoulement de la saumure.

En aval des soutes St1 et St2, un ensemble de 6 vannes (V2VM03, V2VM04, V2VM05, V2VM06, V2VM07 et V2VM08) permet le refoulement du fluide pompé par les trois groupes hydrophores (H1, H2 et H3). Un groupe hydrophore est un dispositif qui permet de maintenir la pression dans le réseau.

Chaque groupe hydrophore est équipé en sortie d'un clapet anti-retour empêchant l'eau de revenir par ce tuyau (Cl5, Cl6 et Cl7). Un ensemble de 3 vannes à trois voies (V3VM01, V3VM02 et V3VM03) permet de choisir, pour chaque hydrophore, si l'eau pompée est envoyée à la distribution (vers Cl9) ou rebouclée sur la ligne de remplissage des soutes (vers Cl8). En aval du clapet (Cl9), le dispositif de stérilisation UV permet le traitement de l'eau en distribution. Ce dispositif est lié à un échangeur thermique (E1) et deux chauffe-eau (CE1 et CE2). L'échangeur thermique permet de refroidir l'eau douce du système par de l'eau réfrigérée fournie par l'interface Int3 isolée de l'échangeur par une vanne de régulation V2VR01 qui permet de réguler le débit d'eau froide. Les pompes PC1 et PC2 connectées au chauffe-eau (CE1) permettent d'assurer la disponibilité de l'eau chaude dans le circuit.

La vanne V2VM09 est utilisée lors de l'alimentation directe du réseau de distribution par le quai. Dans toute autre configuration, cette vanne est fermée.

Comme certains systèmes de contrôle-commande (Bovell, Carter, et Beck 1998), le système EdS dispose de plusieurs éléments redondants qui doivent être activés automatiquement, en cas d'aléas, pour continuer à assurer les fonctions du système, ce qui montre son caractère reconfigurable.

L'analyse des objectifs fonctionnels du système EdS réalisée dans (Bignon 2012) a permis d'identifier sept (7) fonctions de haut-niveau principales : *Production*, *Brassage*, *Distribution*, *Distribution à quai*, *Embarquement*, *Débarquement* et *Transfert*.

La fonction de *production* permet de pomper l'eau de mer pour la traiter par osmose inverse au moyen des osmoseurs afin de produire de l'eau douce qui peut être stockée dans les soutes (St1 et St2), et de l'eau déminéralisée en cas de besoin pour les moteurs du navire.

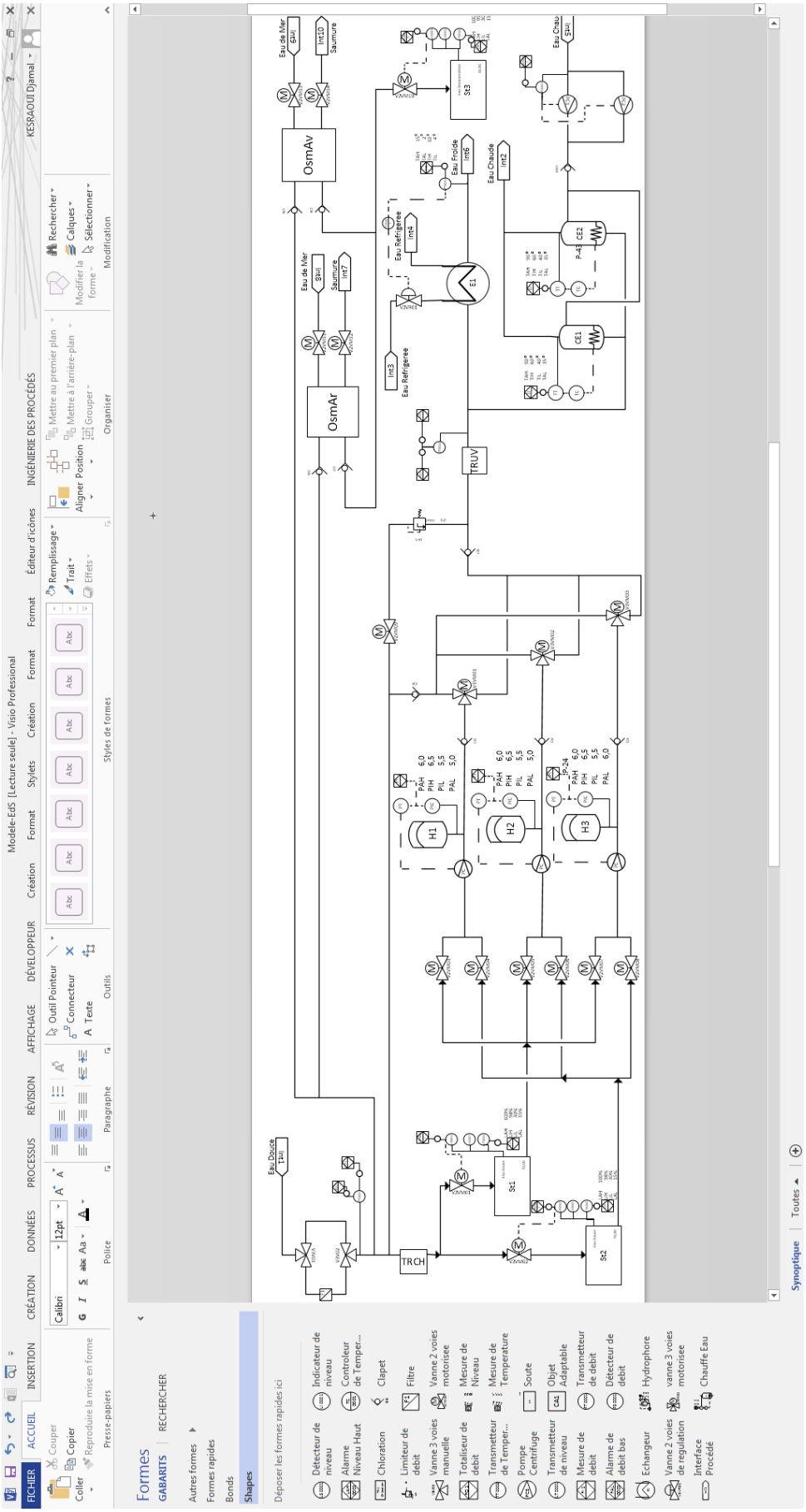


Figure 109 Schéma PI&D du système EdS

L'eau déminéralisée, quand elle est produite, est stockée dans la soute St3. Cette fonction exploite les osmoseurs (OsmAv et OsmAr) ainsi que toutes les vannes des circuits de tuyauterie qui sont liées à la ou les soutes dans lesquelles produire l'eau.

La fonction de *production* permet de pomper l'eau de mer pour la traiter par osmose inverse au moyen des osmoseurs afin de produire de l'eau douce qui peut être stockée dans les soutes (St1 et St2), et de l'eau déminéralisée en cas de besoin pour les moteurs du navire. L'eau déminéralisée, quand elle est produite, est stockée dans la soute St3. Cette fonction exploite les osmoseurs (OsmAv et OsmAr) ainsi que toutes les vannes des circuits de tuyauterie qui sont liées à la ou les soutes dans lesquelles produire l'eau.

Lorsque le navire est à quai, la fonction d'*embarquement* sert à assurer le besoin en eau à bord du navire. Pour réaliser cette fonction, le système est raccordé au port, puis les vannes en amont des soutes (St1 et St2) sont mises en œuvre.

La fonction de *débarquement* est réalisée pour assurer la maintenance des soutes (St1 et St2). Il s'agit de vider l'eau des soutes vers le port (Int1) en exploitant les hydrophores (H1, H2 et H3) ainsi que les vannes du circuit liées entre les soutes et l'élément Int1.

La fonction de *brassage* permet de traiter l'eau douce stockée dans une soute en respectant les normes imposées pour le traitement des eaux potables. Il s'agit de faire passer l'eau contenue dans une soute par le module de chloration pour la traiter, puis la stocker dans cette même soute. Cette fonction exploite les hydrophores (H1, H2 et H3) ainsi que toutes les vannes des circuits de tuyauterie qui sont liées à la soute à brasser.

La fonction de *distribution* consiste à pomper l'eau d'une des soutes St1 ou St2 vers les éléments Int2 et Int6 afin d'assurer la disponibilité de l'eau chaude (Int2) et de l'eau froide (Int6) dans le navire. Cette fonction exploite les hydrophores (H1, H2 et H3) ainsi que toutes les vannes des circuits de tuyauterie qui sont liées à la soute à partir de laquelle l'eau est distribuée.

La *distribution* peut être également effectuée directement à partir du *quai* en raccordant le système au réseau du port et en mettant en œuvre la vanne V2VM09. Ainsi, l'eau filtrée par F1, passe par la vanne V2VM09 pour alimenter le navire en eau chaude par Int2 et en eau froide par Int6.

La fonction de *transfert* est mise en œuvre lorsque l'opération en supervision a besoin de transférer de l'eau d'une soute à l'autre pour assurer la maintenance sur l'une des soutes. Le transfert d'eau s'effectue entre les soutes St1 et St2. Il met en œuvre les hydrophores (H1, H2 et H3) ainsi que les vannes du circuit entre les deux soutes.

Pour chacune de ces 7 fonctions qui peuvent être réalisées manuellement et/ou automatiquement, une commande de haut-niveau doit être implémentée. Les commandes de haut-niveau permettent d'assurer le mode d'exploitation automatique ou semi-automatique, selon si l'opérateur en supervision doit intervenir partiellement ou totalement dans le lancement de la fonction. Malgré l'implémentation des commandes de haut-niveau, l'opérateur garde néanmoins la possibilité de réaliser ces fonctions manuellement, en agissant un à un sur les éléments du système.

L'implémentation de chaque commande de haut-niveau nécessite la définition des spécifications que ce soit du point de vue de l'utilisateur (modèle de tâches) ou du point de vue du système (spécification fonctionnelle).

Dans l'industrie navale, les modèles de tâches ne sont pas utilisés pour ce type de système. Généralement l'expression des spécifications fonctionnelles est faite par l'expert métier au travers d'un tableau (Excel) non normalisé de configurations associé au schéma PID du système qu'il a conçu. Ce tableau de configurations est un document de spécification écrit en langage naturel.

Compte-tenu de l'architecture du système EdS et de son caractère reconfigurable, chaque fonction peut être réalisée suivant plusieurs configurations pour un total de 73 configurations unitaires à définir dans le tableau Excel. Ces configurations ne tiennent pas encore compte de la possibilité d'effectuer plusieurs fonctions simultanément. Charge à l'expert métier de définir ces 73 configurations ainsi que les cas d'exécutions simultanées.

Afin de valider la méthode et les outils proposés pour faciliter la conception des modèles de tâches, la description des spécifications fonctionnelles et l'implémentation des commandes de haut-niveau, nous illustrons dans cette section l'application de la démarche au système EdS précédemment décrit.

1.2 Opération d'adaptation de pattern de tâches en modèles de tâches

Pour la mise en œuvre de la transformation d'adaptation, nous avons défini l'outil Prototask Editor lié au formalisme K-MAD qui permet au concepteur, sans aucune notion de la notation de tâches K-MAD, de lire, parcourir et adapter facilement le pattern de tâches issu de la première opération. Ainsi, Prototask Editor permet de rendre l'adaptation du pattern accessible à un expert non spécialiste des modèles de tâches afin de capturer plus facilement, et de façon plus formelle, les connaissances fonctionnelles basiques du système qu'il conçoit.

Au lancement de Prototask Editor, l'interface affiche le pattern de tâches chargé dans l'outil (Figure 110) de façon progressive, permettant à l'utilisateur de l'adapter. En effet, l'affichage progressif des tâches s'effectue comme dans les simulateurs de tâches. La simulation et l'adaptation sont donc étroitement liées.

L'interface proposée est composée de quatre zones. La principale (*zone 1* Figure 110) affiche le pattern de tâches de façon progressif comme dans l'outil de simulation de tâches Prototask. Elle contient des widgets permettant de modifier le nom d'une tâche active, de supprimer une tâche active, ou d'ajouter une sous-tâche. La *zone 2* affiche une description de la tâche active telle que écrite dans le pattern de tâche. L'opérateur peut écrire ou modifier cette description dans le champ proposé à cet effet. La *zone 3* regroupe tous les préconditions définies lors de la création du pattern de tâches. Enfin, la *zone 4* affiche l'historique, au fur-et-à-mesure que l'utilisateur descend dans l'arbre de tâches.

Nous décrivons ici quelques étapes du processus d'adaptation afin de présenter les différents widgets de Prototask Editor et leur utilité. Sur la Figure 110, nous avons simulé le pattern de tâches jusqu'à la tâche « *Manipulation des commandes prévues* » pour illustrer l'adaptation sur la conception du modèle de tâches pour la fonction de *transfert*. Les widgets « *crayon* » pour modifier et « *poubelle* » pour supprimer sont visibles sur les tâches réalisables. Le widget « modifier » permet à l'utilisateur de modifier le nom de tâche s'il doit l'être. Le widget « *supprimer* » permet de supprimer une tâche qui n'a pas place dans le modèle de tâches qu'il conçoit. Il est important de noter que la suppression d'une tâche, supprime toutes les sous-tâches qui lui sont liées.

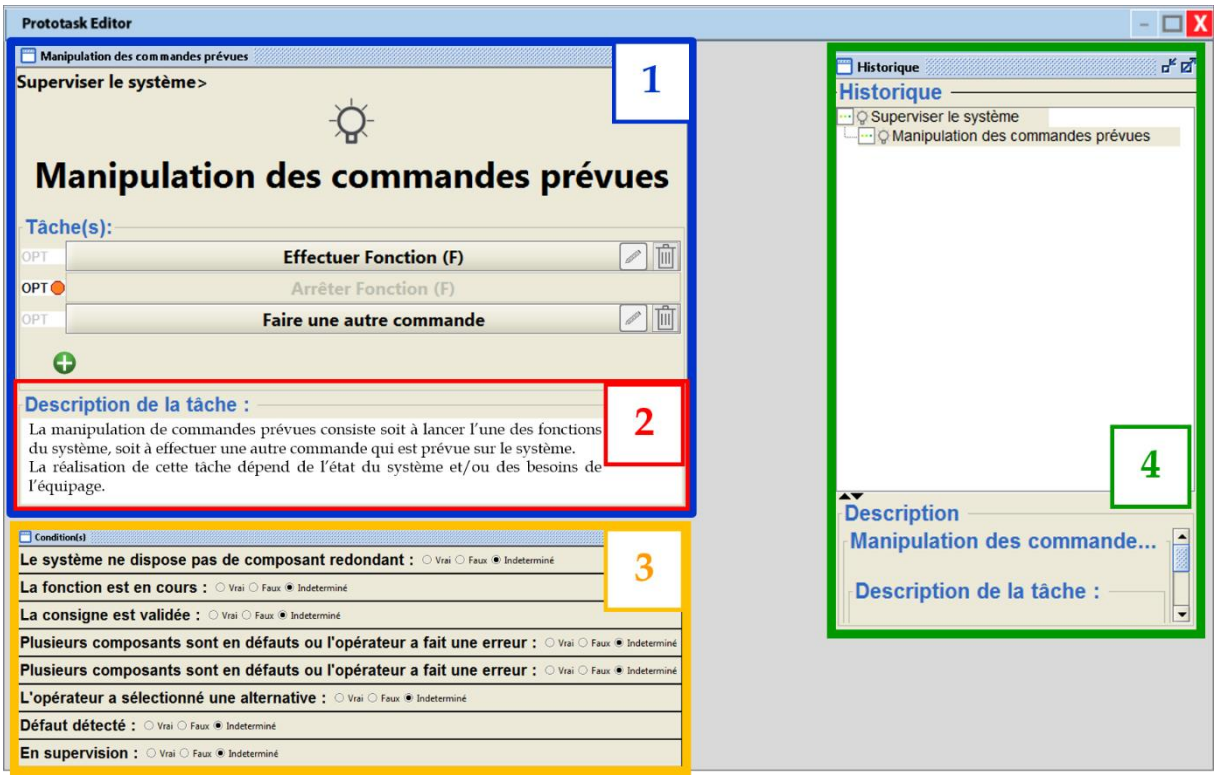


Figure 110 Prototask Editor

Sur la Figure 111, le nom de la tâche « Effectuer Fonction (F) » de la Figure 110 est remplacée par le nom « Effectuer Transfert ». Pour ce faire, l'utilisateur clique sur le widget « modifier » pour renseigner le nouveau nom de tâche, puis valide. La validation du nom permet d'une part de prendre en compte le texte renseigné par l'utilisateur et, d'autre part, de simuler la tâche en question. On retrouve donc dans le volet « Historique » de l'interface (Figure 111) que la tâche « *Effectuer Transfert* » a été réalisée. La zone 1 de l'interface affiche maintenant les sous-tâches de la tâche « *Effectuer Transfert* » pour permettre à l'utilisateur de continuer la simulation et l'adaptation de son modèle de tâches.

Sur la Figure 112, l'utilisateur expert est descendu dans le modèle jusqu'au niveau « Renseigner consigne » en modifiant le nom des tâches, s'il y a besoin. Il remplace les noms de tâches :

- « *Lancer Fonction (F)* » par « *Lancer Transfert* »,
- « *Renseigner (n) consigne* » par « *Renseigner la consigne* »,
- « *Consigne (n1)* » par « *Soute de départ* »,
- « *Consigne (nx)* » par « *Soute d'arrivée* ».

L'historique permet de retracer les actions de l'utilisateur expert. Ainsi, l'utilisateur expert peut visualiser ses actions et le niveau où il se trouve dans le pattern de tâches instancié.

L'utilisateur expert peut avoir besoin d'ajouter une sous-tâche. Par exemple, sur la Figure 112, il a adapté deux consignes pour la fonction de transfert et remarque qu'il lui faut renseigner une consigne. Il clique alors sur le widget « Ajouter » pour ajouter autant de tâches que nécessite la fonction dont il conçoit le modèle.

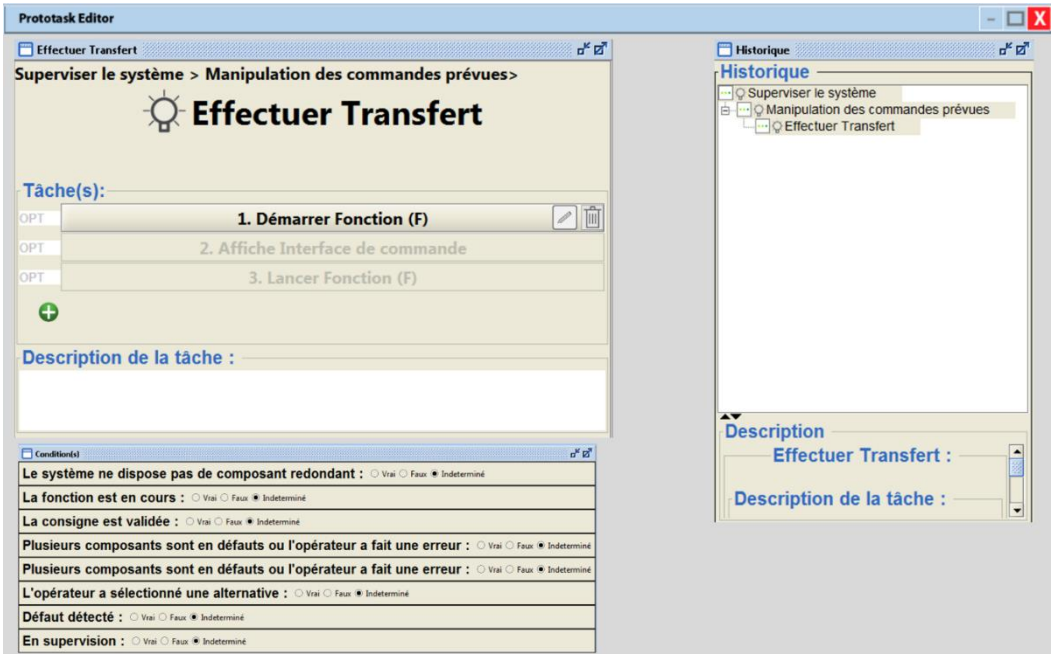


Figure 111 Prototask Editor – Changement de nom d'une tâche

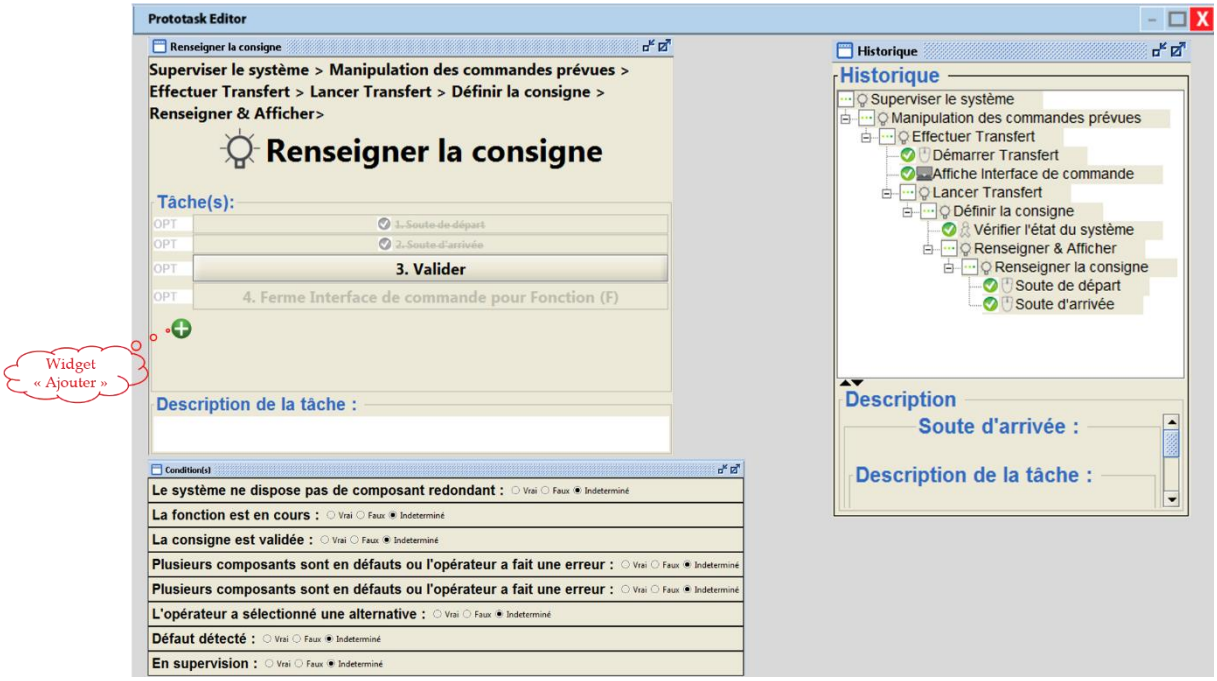


Figure 112 Prototask Editor – illustration du widget « ajouter »

L'adaptation porte essentiellement sur le nom des tâches, la suppression des tâches nx s'il y en a trop, l'ajout d'une tâche sous l'arbre s'il n'y en a pas assez, par exemple pour définir une consigne.

À la fin du processus d'adaptation on obtient le modèle de tâches de la fonction de Transfert (Annexe 3).

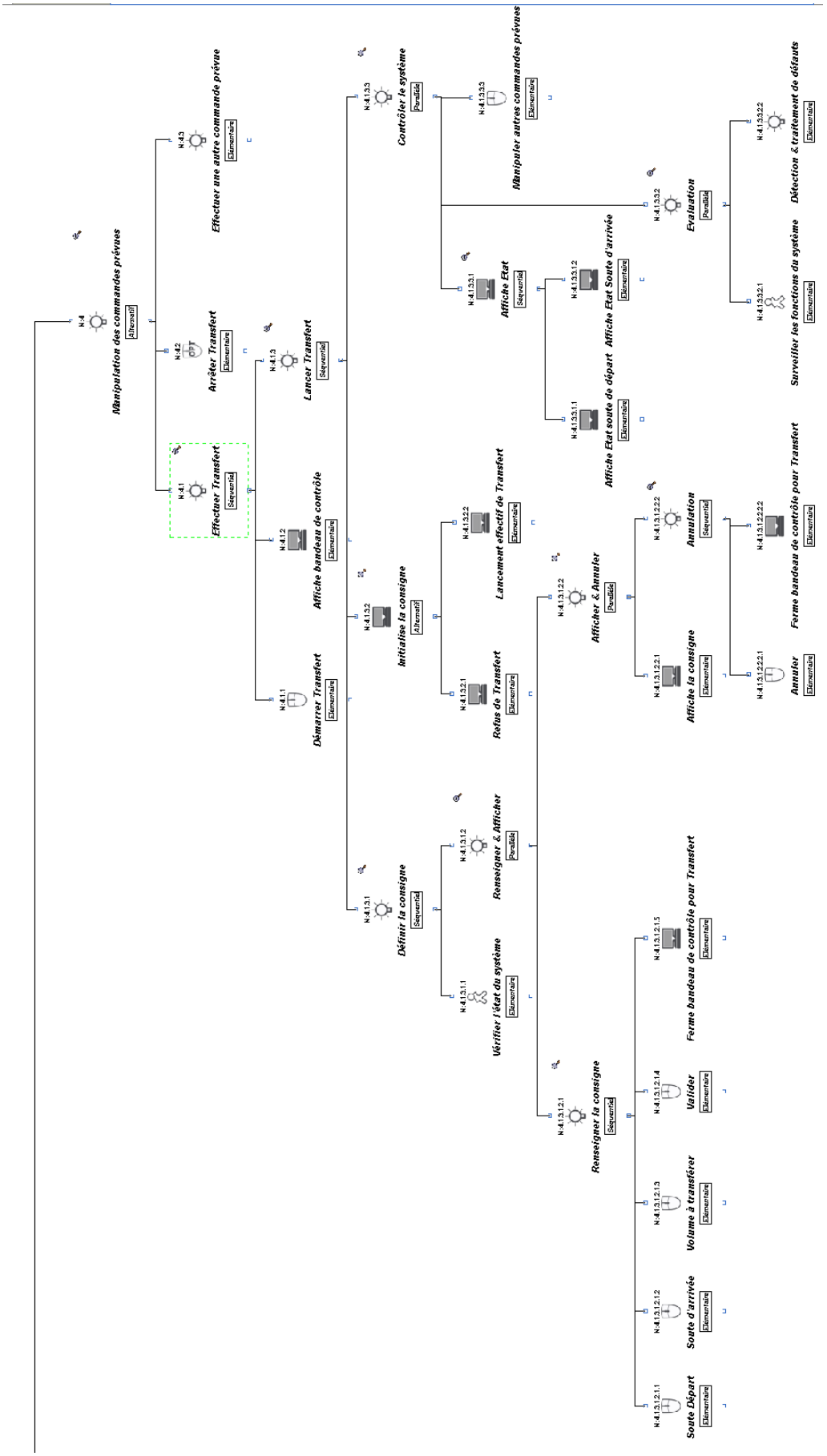


Figure 113 Modèle de tâches de la fonction de transfert

Le modèle de tâches de chaque fonction décrit les actions que l'opérateur (marin) en supervision fait pour lancer une fonction, contrôler et surveiller les évènements du système (les alertes, les messages de bon fonctionnement), évaluer et arrêter la fonction (s'il y a besoin). Nous prenons l'exemple du modèle de tâches de Transfert pour l'expliquer. La Figure 113 présente un extrait du modèle de tâches de cette fonction. Selon l'état du système, l'opérateur peut, soit faire la tâche « *Effectuer Transfert* » si elle n'est pas en cours, soit décider d' « *Arrêter Transfert* » en cours. Ces deux tâches sont alternatives et sont pré-conditionnées (Figure 113). Lorsque l'opérateur décide par exemple de faire la tâche « *Effectuer Transfert* », il clique sur la commande de haut-niveau « *Démarrer Transfert* ». Le système « *Affiche bandeau de contrôle* » lié à cette commande de haut-niveau. Pour lancer le transfert, l'opérateur doit vérifier l'état des éléments, renseigner une consigne, puis valider la consigne ou l'annuler. Le renseignement de la consigne se fait sur le bandeau de contrôle. Le système affiche les actions à faire sur ce bandeau, au fur-et-à-mesure que l'opérateur réalise les tâches liées à la consigne. Dans le cas où l'opérateur valide la consigne, le système l'initialise, puis commence une phase de contrôle. Pendant cette phase, l'opérateur doit superviser et agir en fonction des retours du système.

Selon (Johnson et al. 1993), les modèles de tâches ont été reconnus comme un élément important dans la conception de l'interface utilisateur car ils contiennent la connaissance des intentions et des activités de l'utilisateur. Dans notre cas, on voit bien que toutes ces informations sont bien décrites au niveau du modèle de tâches. Ces modèles peuvent bien servir de point de départ pour la génération des interfaces des commandes de haut-niveau qui permettront à l'opérateur en supervision de piloter l'IHM.

Pour avoir tous les modèles de tâches des 6 autres fonctions du système, le même processus d'adaptation est utilisé. Les modèles de tâches simulés, vérifiés et validés, tout en rendant l'arbre de tâches complètement transparent à l'utilisateur expert qui n'a aucune connaissance des notations de tâches.

1.3 Opération d'adaptation du modèle EGRC

L'exécution des transformations d'épuration, d'identification et de mise à jour du modèle ER nous renvoie un modèle EGRC contenant les exigences fonctionnelles de bas-niveau permettant de faciliter la spécification fonctionnelle du système. L'adaptation porte uniquement sur la partie fonctionnelle du modèle EGRC, la présentation n'est pas concernée. Notons que cette phase est entièrement automatique.

Dans le modèle EGRC, il existe un script générique (Figure 114) qui permet de récupérer la liste des fonctions du système à concevoir, issue de l'étape d'identification. La variable « *ListeDesFonctions* » contient, pour le système EdS, 7 fonctions identifiées dans le cahier de charges.

```
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Repertoire = PnGetValue("../#Parent.Repertory")
Set FSO = CreateObject("Scripting.FileSystemObject")
'liee la liste des fonctions a l enregistreur
chemin = Repertoire & "ListeDesFonctions.txt"
    'ouverture du fichier en lecture
Set F = fso.OpenTextFile(chemin, ForReading, True)
    'lecture du fichier
Variable = F.ReadAll
'mettre le contenu du fichier dans la variable ListeDesFonctions
PnSetValue "../#Parent.ListeDesFonctions", Variable
F.Close
```

Figure 114 Récupération de la liste de fonctions issue de l'étape d'identification

Les modèles de tâches vérifiés et validés sont épurés pour ne récupérer que les noms et les descriptions des sous-tâches interactives et systèmes élémentaires de la tâche « Renseigner consigne » et « *Afficher les alertes* ». L'objectif de cette adaptation est d'utiliser les exigences fonctionnelles recueillies par les modèles de tâches pour guider l'utilisateur expert lors de l'utilisation du modèle EGRC.

La Figure 115 montre un extrait de l'adaptation du modèle EGRC. Ici, les noms des tâches interactives élémentaires ainsi que l'opérateur temporel de leur tâche mère sont utilisées pour adapter le modèle EGRC. D'une part, les noms des tâches sont utilisés pour guider l'utilisateur expert lors de la spécification du système en mettant le mot « Renseignez » au début de chaque nom de tâche pour créer les différentes variables permettant le guidage de l'utilisateur expert (par exemple *GuidageZonText1* sur Figure 115).

D'autre part, le nom des tâches est utilisé pour créer les variables d'affichage qui confirment la tâche que l'opérateur vient de réaliser (par exemple, *AffichageZonText1* sur Figure 115).

Par ailleurs, l'ordre d'affichage des messages de guidage est contrôlé par les informations issues de l'opérateur temporel de la tâche mère dont les sous-tâches sont utilisées. Ces informations sont stockées dans les variables de visibilité (par exemple *BeingSetZonText1* sur Figure 115).

```
PnSetValue "../#Parent.GuidageZonText1", "Renseignez Soute de départ "
PnSetValue "../#Parent.GuidageZonText2", "Renseignez soute d'arrivée "
PnSetValue "../#Parent.GuidageZonText3", "Renseignez Volume à transférer "
PnSetValue "../#Parent.AffichageZonText1", "Soute de départ :"
PnSetValue "../#Parent.AffichageZonText2", "Soute d'arrivée :"
PnSetValue "../#Parent.AffichageZonText3", "Volume à transférer :"
PnSetValue "../#Parent.BeingSetZonText1", True
```

Figure 115 Exemple d'adaptation du composant EGRC à partir des données issues des modèles de tâches pour guider l'utilisateur expert lors de spécification

On retrouve dans le modèles EGRC après adaptation, les informations relatives au renseignement des consignes et aux conditions d'arrêt décrites dans les modèles de tâches. Par exemple, sur la Figure 116, à la variable *ListeConditionsFin* est associée la condition de fin de chaque fonction par le concepteur lors de l'adaptation du modèle de tâches.

```
Select Case NomFonction
    Case "Transfert"
        PnSetValue "../#Parent.ListeConditionsFin", "VolumeTransfert atteint;NiveauBas de la Soute de départ;NiveauHaut de la soute d'arrivée;Defaut d'un élément"
    Case "Brassage"
        PnSetValue "../#Parent.ListeConditionsFin", "Volume Soute de Brassage atteint"
    Case "Debarquement"
        PnSetValue "../#Parent.ListeConditionsFin", "NiveauBas de la soute de débarquement"
    Case "Embarquement"
        PnSetValue "../#Parent.ListeConditionsFin", "NiveauHaut de la soute d'embarquement"
    Case "Distribution"
        PnSetValue "../#Parent.ListeConditionsFin", "NiveauHaut de la pression de l'hydrophore"
    Case "Production"
        PnSetValue "../#Parent.ListeConditionsFin", "NiveauHaut de la soute de production"
End Select
```

Figure 116 Exemple d'adaptation du composant EGRC – Introduction des données relatives aux alertes de chaque fonction

La description des tâches utilisées, renseignée par l'utilisateur expert lors de l'adaptation des modèles de tâches, est également introduite dans le widget « *Aide* » qui se trouve aux différents endroits de l'enregistreur. Lorsque l'utilisateur expert clique sur ce widget, il peut lire cette description et bien réaliser la tâche à faire.

1.4 Opération d'insertion Spec

L'IHM d'ERGC (Figure 117) est obtenue par génération automatique après la réalisation de l'opération d'insertion Spec. Elle est cohérente avec le schéma P&ID de la Figure 109 construit par l'expert et contient les widgets du modèle EGRC précédemment adapté. Ces widgets

permettent de mettre le système en mode enregistrement pour enregistrer les exemples de spécifications décrites par le concepteur, en mode rejeu pour rejouer les spécifications proposées par le système, ou en mode paramètre pour modifier les vues des commandes globales générées. Pour ce faire, ces widgets communiquent avec les éléments du système.

Les éléments de l'IHM de spécification visibles par le concepteur prennent en compte ces commandes de bas-niveau (consigne exprimée sur un élément du procédé représenté sur l'IHM) et lui fait un retour d'information sur l'état du composant sur lequel il vient d'interagir. Autrement dit, pendant la phase de spécification, le concepteur perçoit les retours du système comme ce sera le cas en temps réel. En effet, l'animation de l'interface est gérée par Panorama E2 au travers d'équations générées par Anaxagore. À chaque nouvel état de l'interface, le moteur de Panorama E2 recalcule les équations, et adapte la visualisation en fonction de leurs résultats. L'utilisateur peut donc commander chacun des composants du système présents sur l'IHM. À partir de ces actions sur l'IHM, les changements d'état des variables internes à Panorama permet à l'utilisateur d'observer la réaction du système réel par rapport à son action sur celui-ci et de décrire les bonnes séquences d'actions pour l'enregistrement des spécifications.

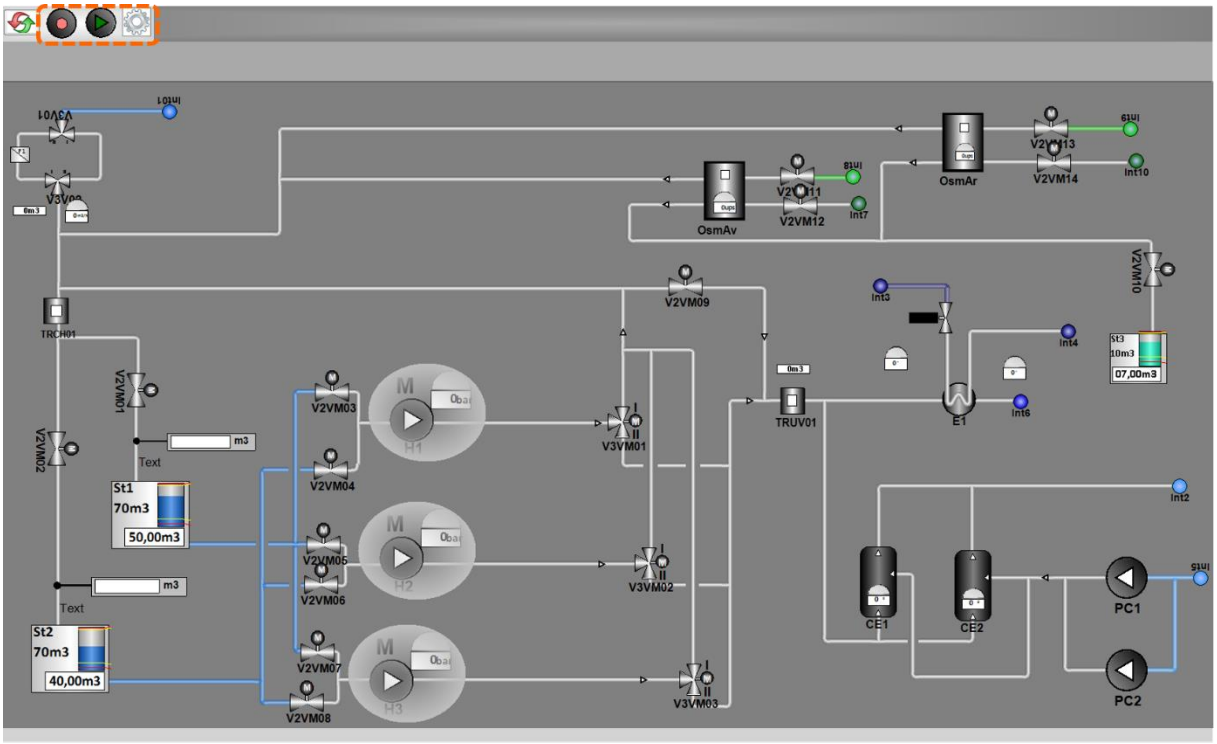


Figure 117 Résultat de l'opération d'insertion Spec – IHM d'EGRC

Ainsi, l'IHM de spécification permet notamment de décrire quelques exemples de scénarios des fonctions de haut-niveau du système. Dans le logiciel Panorama E2, elle se compose d'un nombre important d'objets de base du logiciel, que l'on peut assimiler à des widgets généralement connectés à un programme de contrôle-commande. Le principe des widgets permet à chaque composant d'encapsuler son comportement, la dynamique de l'animation nécessaire à la mise en évidence des phénomènes physiques du système.

Par exemple, sur la Figure 117, toutes les vannes sont fermées : l'eau ne passe donc qu'entre les soutes et les premières vannes. Les tuyaux en bleu sont remplis d'eau et ceux en gris sont vides. Pour faire passer l'eau, le concepteur doit ouvrir une vanne. Lorsqu'il clique sur une

vanne pour l'ouvrir, un menu circulaire s'affiche avec les boutons lui offrant la possibilité d'agir sur le composant. Lorsqu'il clique sur le bouton Ouverture, la commande associée à ce bouton envoie au système physique un ordre de changement d'état. La variable correspondant dans l'API change alors d'état et est transmise à la supervision pour montrer à l'utilisateur que la vanne est ouverte. Dans ce cas, le concepteur observe les animations : l'intérieur de la vanne passe en bleu foncé (il y a de l'eau qui traverse la vanne) ainsi que les tuyaux en aval de la vanne.

La couleur du barre-graphe de la soute dépend de la nature du fluide stocké. Un code couleur a été défini sur la base de l'analyse des résultats de tests utilisateurs réalisée dans le cadre de la colorimétrie (Boulhic, Bignon, et Silone 2016). Par exemple, le bleu correspond à de l'eau douce.

1.5 Opération d'enregistrement - généralisation

La démarche d'enregistrement–généralisation est mise en œuvre suivant deux phases. Pendant la première phase, l'utilisateur expert utilise l'IHM d'EG pour montrer des exemples de séquences d'exécution pour toutes les fonctions (exemples de spécifications fonctionnelles). Deux catégories de généralisation sont faites au cours de la deuxième phase : la généralisation du programme et la généralisation des configurations.

Afin de vérifier la faisabilité de notre approche, nous l'appliquons à la définition de la spécification des fonctions du système EdS. L'IHM d'EG (Figure 117) générée est interprétable par le logiciel de type SCADA Panorama E2. Sa génération se base sur un modèle d'EGRC adapté à partir des informations issues de modèles de tâches, et sur les vues graphiques des éléments du système à concevoir, stockées dans des bibliothèques d'éléments. Les widgets Enregistreur/Rejoueur/Correcteur se trouvant en haut à gauche de l'IHM (Figure 117), permettent la mise en œuvre des techniques d'EUD. Le modèle de tâches de chaque fonction décrit les actions que l'opérateur en supervision effectue pour lancer, contrôler, surveiller les événements du système (les alertes, les messages de bon fonctionnement), évaluer et arrêter le transfert (s'il y a besoin). Par exemple, comme décrit dans le modèle de tâches de la fonction de Transfert, pour effectuer cette fonction, l'opérateur en supervision doit renseigner, puis valider une consigne. Ces informations servent à guider l'utilisateur expert par rapport aux actions à réaliser tout en respectant l'ordre décrit dans le modèle de tâches, lors de la spécification des exemples de cette fonction.

1.5.1 L'enregistrement

Lors de l'*enregistrement*, le système enregistre les départs et arrivées possibles pour la fonction et toute la séquence d'actions faite par l'utilisateur expert ainsi que l'état des éléments manipulés pour la réalisation des fonctions (les configurations). Pour être généralisés, deux exemples de chaque fonction doivent être enregistrés (Figure 118 et Figure 119).

Transfert1.vbs

```
1  PnSetValue"../SouteDepart.Value", St1
2  PnSetValue"../SouteArrivee.Value", St2
3  PnSetValue"../Volume.Value", 5
4  PnSetValue"../ValidationTransfert.Value", 1
5  Res = PnWaitProperty("../VolumeCompteur_St2", "=",
6  "../VolumeTransfert", 0)
7  PnSetValue"../ValidationTransfert.Value", 0
```

Figure 118 Exemple1 de programme enregistré pour la fonction de transfert

Transfert2.vbs

```
1   PnSetValue"../SouteDepart.Value", St2
2   PnSetValue"../SouteArrivee.Value", St1
3   PnSetValue"../Volume.Value", 10
4   PnSetValue"../ValidationTransfert.Value", 1
5   Res = PnWaitProperty("../VolumeCompteur_St1",
6   "=", "../VolumeTransfert", 0)
7   PnSetValue"../ValidationTransfert.Value", 0
```

Figure 119 Exemple2 de programme enregistré pour la fonction de transfert

La séquence d'actions faites par l'utilisateur lors de l'enregistrement des exemples se réalise suivant les tâches décrites dans les modèles de tâches de fonction. Lors de la réalisation de ces tâches, le système renvoie un *modèle d'interaction et des modèles de dialogue.* Les modèles de dialogues décrivent la réaction du système aux actions de l'utilisateur, relative à la même tâche.

Nous illustrons ici la structure des modèles d'interaction et de dialogues enregistrés pour la fonction de transfert :

- **Tâche1 : Renseignez Soute de départ**

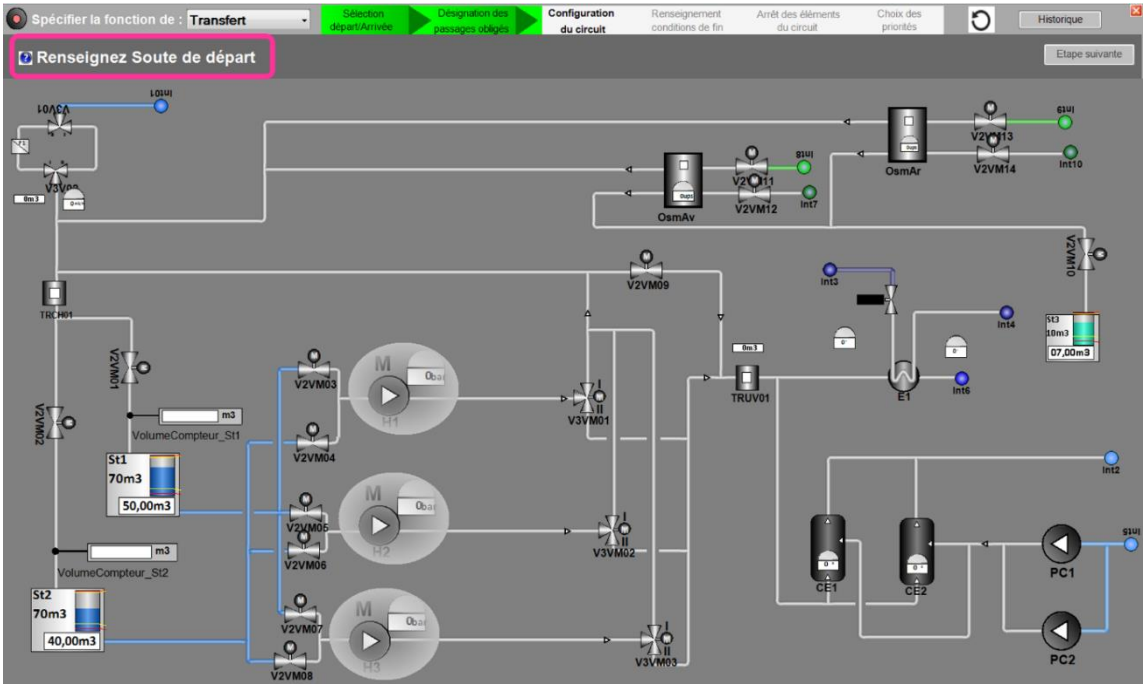
Dans un premier temps, le système propose à l'expert de réaliser une tâche « *Cliquez sur la soute de départ* » (Figure 121a). Pour réaliser cette tâche, l'expert a cliqué sur l'un des éléments de l'interface pour renseigner la soute de Départ (Figure 121b). Il a donc sélectionné un élément parmi les *n* éléments affichés sur l'interface. Une règle est écrite pour déduire que cette sélection correspond à un choix parmi *n,* et que l'objet d'interaction le mieux adapté pour la réalisation de cette tâche (en dehors de l'interface) est une *liste déroulante (drop-down List)*.

Lorsque l'utilisateur valide son choix (Figure 121b), le système prend en compte le choix et passe à la tâche suivante. Cette réaction du système est enregistrée comme dialogue intra-fenêtre (Figure 120) et le lien vers ce programme est enregistré dans le modèle de dialogue lié à la fonction.

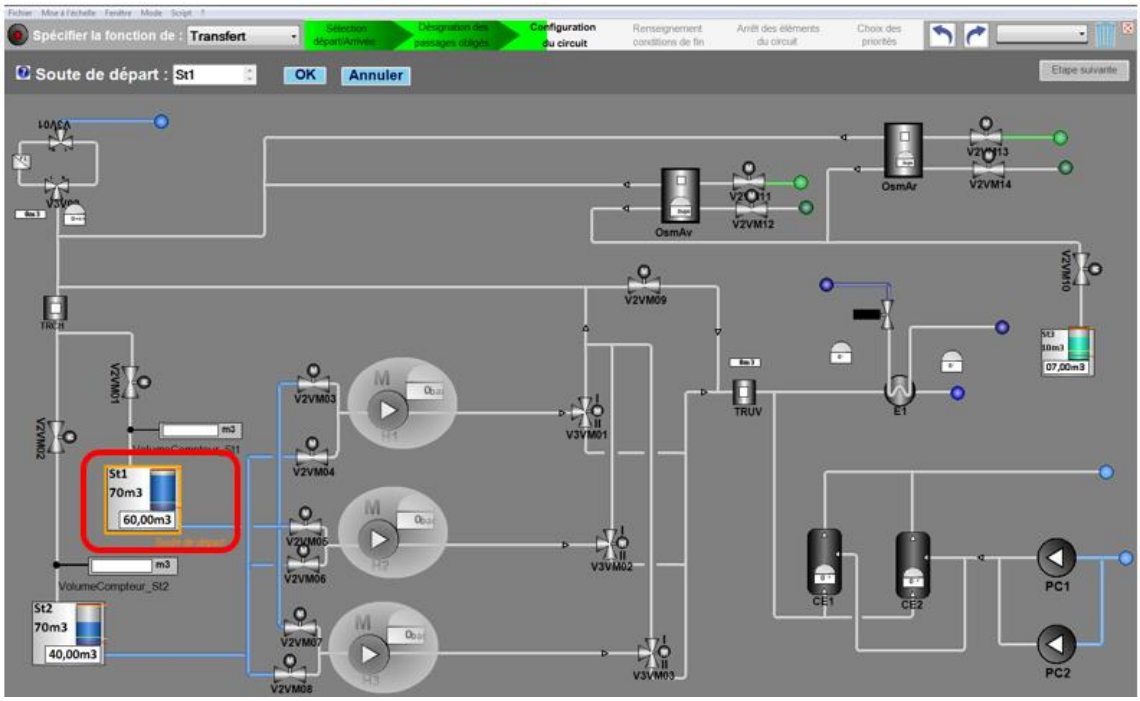
Transfert1.vbs

```
1   PnSetValue "../SouteDepart.Value", SouteDepart.Value
2   PnSetValue("Consign2.Value") , True
```

Figure 120 Exemple de dialogue intra-fenêtre pour la tâche 1 de la fonction de transfert



(a)



(b)

Figure 121 Déduction de widget d'interaction pour le choix du départ. (a) La tâche à effectuer issue du modèle de tâches. (b) L'action faite par l'expert pour effectuer la tâche en question.

Pour chaque tâche, les informations relatives aux modèles d'interaction et de dialogues ainsi que la réponse du système sont enregistrés en plus de la séquence d'actions réalisée par l'utilisateur. La Figure 122 présente la structure du modèle de dialogue. On y retrouve les dialogues intra-fenêtre mais aussi les liens vers les intra-widget. Pour la fonction de transfert, les dialogues intra-widget sont par exemple les liens vers les ensembles de départs/arrivées renseignés lors de la première étape de spécification.

DialogModel.xml

```
<?xml version="1.0" encoding="utf-8"?>
<xmi:DialogModel xmlns:xmi="http://www.segula.fr/Anaxagore/CheckingValidation/DialogModel">
<DialogLinkList FunctionName="Transfert">
    <DialogLink id="1">
        <Task taskname="Démarrer Transfert"></Task>
    </DialogLink>
    <DialogLink id="2">
        <Task taskname="Soute de départ"></Task>
        <IntraWidScriptLink ScriptLink="Fonctions\ElementsEntreeDepart\Transfert.txt"></IntraWidScriptLink>
        <IntraWinScriptLink ScriptLink="ModelesDialogue_Lancement\Transfert\Transfert1.vbs"></IntraWinScriptLink>
    </DialogLink>
    <DialogLink id="3">
        <Task taskname="Soute d'arrivée"></Task>
        <IntraWidScriptLink ScriptLink="Fonctions\ElementsEntreeArrivee\Transfert.txt"></IntraWidScriptLink>
        <IntraWinScriptLink ScriptLink="ModelesDialogue_Lancement\Transfert\Transfert2.vbs"></IntraWinScriptLink>
    </DialogLink>
    <DialogLink id="4">
        <Task taskname="Entrer le volume à transférer"></Task>
        <IntraWinScriptLink ScriptLink="ModelesDialogue_Lancement\Transfert\Transfert3.vbs"></IntraWinScriptLink>
    </DialogLink>
    <DialogLink id="5">
        <Task taskname="Valider"></Task>
        <IntraWinScriptLink ScriptLink="Generalisations\ScriptsGeneralises\SriptsLancement\Transfert.vbs"></IntraWinScriptLink>
    </DialogLink>
    <DialogLink id="6">
        <Task taskname="Arrêter Transfert"></Task>
        <IntraWinScriptLink ScriptLink="Generalisations\ScriptsGeneralises\SriptsLancement\Transfert_ArretOperateur.vbs"></IntraWinScriptLink>
    </DialogLink>
</DialogLinkList>
```

Figure 122 Structure des dialogues enregistrés

L'extrait présenté sur la Figure 123 décrit la structure du *modèle d'interaction*. Ce modèle contient un widget (suggéré en fonction de l'action faite par l'utilisateur pour réaliser la tâche).

InteractionModel.xml

```
<?xml version="1.0" encoding="utf-8"?>
<xmi:InteractionModel xmlns:xmi="http://www.segula.fr/Anaxagore/
<InteractionList FunctionName="Transfert">
    <Interaction id="1">
        <Task taskname="Démarrer Transfert"></Task>
        <Widget Category="PushButton"></Widget>
    </Interaction>
    <Interaction id="2">
        <Task taskname="Soute de départ"></Task>
        <Widget Category="drop-down list"></Widget>
    </Interaction>
    <Interaction id="3">
        <Task taskname="Soute d'arrivée"></Task>
        <Widget Category="drop-down list"></Widget>
    </Interaction>
    <Interaction id="4">
        <Task taskname="Volume à transférer"></Task>
        <Widget Category="Edit"></Widget>
    </Interaction>
    <Interaction id="5">
        <Task taskname="Valider"></Task>
        <Widget Category="PushButton"></Widget>
    </Interaction>
    <Interaction id="6">
        <Task taskname="Arrêter Transfert"></Task>
        <Widget Category="PushButton"></Widget>
    </Interaction>
</InteractionList>
```

Figure 123. Structure du modèle d'interaction –Exemple Fonction de Transfert

4.1.1 Généralisation

Un système de *généralisation* utilise les deux exemples enregistrés (Figure 118 et Figure 119) pour en déduire un programme générique (Figure 124) prenant en compte les autres configurations possibles d'une fonction.

Transfert.vbs

```
1  PnSetValue"../SouteDepart.Value", SouteDepart.Value
2  PnSetValue"../SouteArrivee.Value", SouteArrivee.Value
3  PnSetValue"../Volume.Value", Volume.Value
4  PnSetValue"../ValidationTransfert.Value", 1
5  Res = PnWaitProperty("../VolumeCompteur_" & SouteArrivee.Value,
6  "=", "../VolumeTransfert", 0)
7  PnSetValue"../ValidationTransfert.Value", 0
```

Figure 124 Résultat de la généralisation des deux exemples de scripts enregistrés

La généralisation des configurations suit plusieurs étapes. Tout d'abord, l'utilisation d'un algorithme de combinaison, décrit en s'appuyant sur l'équation 1 au Chapitre 4, permet de proposer à l'utilisateur tous les chemins possibles qu'un système peut utiliser pour atteindre l'objectif fonctionnel demandé. Ces chemins possibles sont validés par l'expert métier puis combinés avec l'utilisation d'un solveur et d'un algorithme de configuration. Le but est de fournir l'exhaustivité des possibilités de configuration permettant l'exécution des fonctions spécifiées.

La Figure 125 présente le résultat de la généralisation des configurations pour la fonction *Transfert*. En fonction des départs/arrivées renseignés lors de l'enregistrement, le système de généralisation de configurations calculent tous les chemins possibles et les combinaisons possibles de départ/arrivée, puis récupèrent les variables d'états des éléments sur ces chemins.

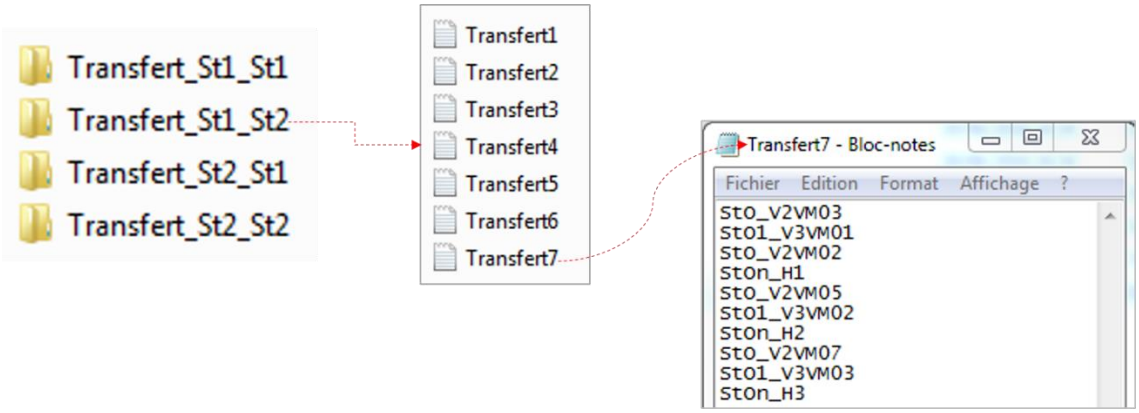


Figure 125 Résultat de la généralisation des configurations

Les configurations généralisées proposées par le système ne sont pas forcément toutes utiles pour la fonction considérée. L'opération de vérification et validation va permettre à l'utilisateur de ne choisir que les configurations nécessaires.

4.2 Opération de génération d'interfaces de contrôle

Les configurations possibles générées, ainsi que les modèles d'interface de fonctions doivent être vérifiées et validées par l'expert lors de cette phase. Il s'agit d'offrir à l'utilisateur qui n'a aucune notion de programmation le moyen de vérifier et valider les spécifications fonctionnelles obtenues (programmes génériques (Figure 124) et configurations généralisées (Figure 125)). La vérification et la validation peuvent se faire à travers les interfaces des contrôle-commandes de haut-niveau (High Level Control-Command Interface (HLCCI)).

Le HLCCI est une interface composée de plusieurs widgets : un widget (de type container) contenant tous les widgets des commandes de haut niveau (Figure 126a) et un widget (de type

display) qui permet d'afficher les informations de haut niveau (Figure 126b). Sur la Figure 126.a, les commandes de démarrage et d'arrêt de chaque fonction sont superposées les unes sur les autres. Les actions à faire pour réaliser une fonction à partir du HLCCI sont décrites dans le modèle de tâches de chaque fonction.

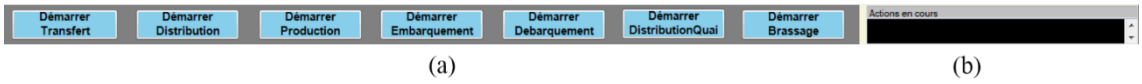


Figure 126 HLCCI

Pour générer les HLCCI, nous complétons les modèles de tâches précédemment obtenus par un modèle de tâches de lancement des contrôles-commandes globaux en parallèle (Figure 127), qui regroupe à un niveau très abstrait les actions que l'utilisateur peut faire pour lancer les fonctions de haut-niveau. Ce modèle de tâches permet de recueillir quelques exigences fonctionnelles sur l'ensemble des fonctions du système. Il permet d'exprimer le fait que l'opérateur (marin) peut réaliser une fonction selon l'état du système. La réalisation de certaines fonctions est pré-conditionnée et dépend de la priorité attribuée à la fonction. En termes de précondition, par exemple, pour lancer la Distribution à quai, il faut que le navire soit à quai. L'opérateur peut lancer une fonction si elle n'est pas en cours ou arrêter une fonction qui est en cours.

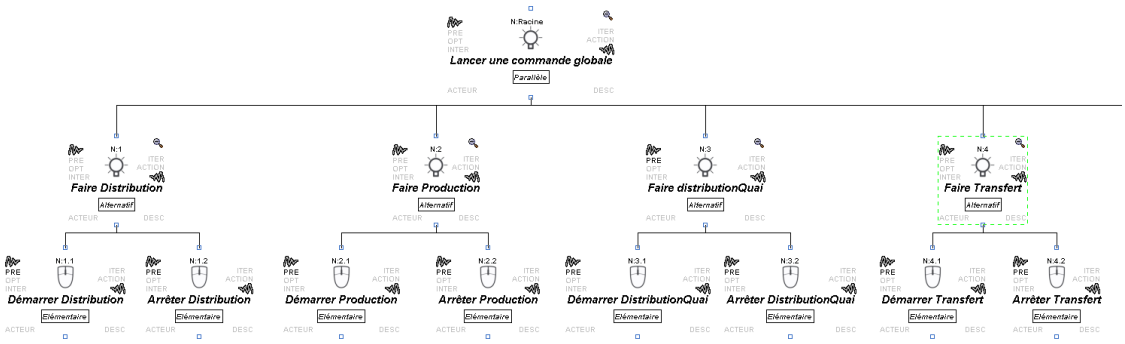


Figure 127. Extrait du modèle de tâches de lancement de contrôle-commandes globaux

La première étape de notre démarche est de générer, à partir de modèles de tâches issus de l'opération de spécification de tâches humaines (voir section 1.2 du chapitre 5), des ensembles de présentations (PTS) et les dialogues inter-fenêtres liés à ces ensembles. Lorsque nous appliquons la démarche décrite dans la section 3.3.1.1 (Chapitre 4, pp. 105) à l'analyse d'un extrait de l'arbre de tâches de la fonction de transfert (la partie qui concerne le lancement de la commande globale), on obtiendra les PTS suivant :

PTS1 :{Démarrer Transfert, Arrêter Transfert}
PTS2 : {Affichage bandeau control}
PTS3 : {Soute Départ, Afficher consigne, Annuler}
PTS4 : {Soute d'arrivée, Afficher consigne, Annuler}
PTS5 : {Entrer volume à transférer, Afficher consigne, Annuler}
PTS6 : {Valider, Afficher consigne, Annuler}
PTS7 : {Fermeture bandeau}

Si nous appliquons l'heuristique *Sharing most elements Sets*, qui indique que certains PTS qui partagent le plus d'éléments sont unifiés, on obtiendra les PTS suivant pour notre exemple :

PTS1 :{Démarrer Transfert, Arrêter Transfert}
PTS2 : {Affichage bandeau control}

PTS3 : {Soute Départ, Afficher consigne, Annuler,
Soute d'arrivée, Entrer volume à transférer, Valider}
PTS4 : {Fermeture bandeau}

Si nous appliquons l'heuristique *Singe Element Sets*, qui indique que, si un PTS est composé de seulement un élément, il est associé à un autre PTS, on obtiendra les PTS suivants pour notre exemple :

PTS1 :{Démarrer Transfert, Arrêter Transfert}
PTS2 : {Soute Départ, Afficher consigne, Annuler,
Soute d'arrivée, Entrer volume à transférer, Valider,
Affichage bandeau control, Fermeture bandeau}

Après l'application de ces deux heuristiques, nous obtenons deux ensembles de présentation pour l'interface de contrôle-commande globale de la fonction de transfert. Ces deux ensembles sont reliés par des transitions exprimées avec des règles en se basant sur la décomposition séquentielle et l'ordre des tâches (Figure 128). Ces transitions représentent le dialogue inter-fenêtre reliant les deux ensembles de présentations obtenus.

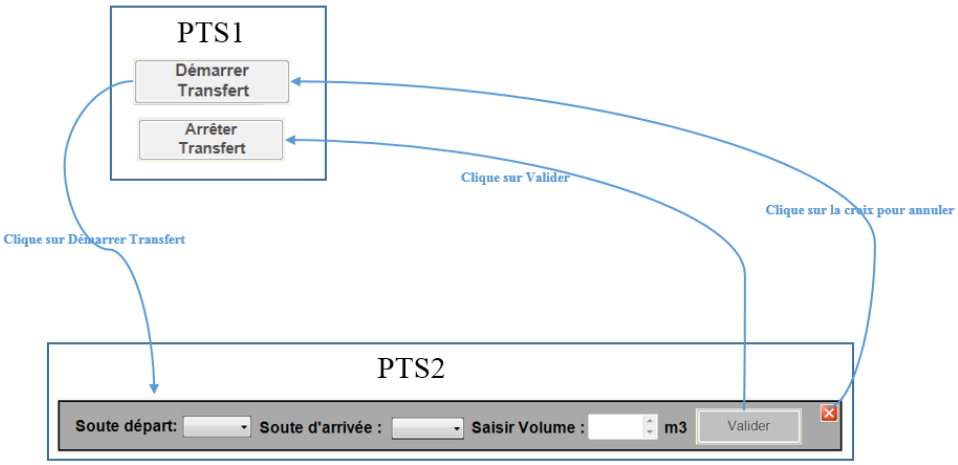


Figure 128. PTS et dialogue inter-fenêtre générés à partir du modèle de tâches de Transfert

La deuxième étape de notre démarche est de collecter les informations relatives au modèle d'interaction (Figure 123) et aux dialogues (Figure 122) enregistrés (par exemple le ***script*** de la Figure 120) et généralisés (par exemple le ***programme générique*** (Figure 124)) lors de l'opération d'enregistrement-généralisation, pour établir la liaison entre les widgets proposés et les modèles de dialogue enregistrés (Figure 86 Chapitre 4). Cette étape nous renvoie à la collection de widgets suggérés, qui contient pour chaque fonction les tâches du modèle de tâches à réaliser, puis les widgets et les liens vers les dialogues enregistrés et/ou généralisés lors de la réalisation de la dite tâche.

Cette collection, les dialogues inter-fenêtres et les PTS sont utilisés au cours de la dernière étape pour générer les interfaces de contrôle-commandes de haut-niveau conformément au méta-modèle de la Figure 91 (pp. 144).

WidgetCollection.xml

```
<?xml version="1.0" encoding="utf-8"?>
<xmi:DialogModel xmlns:xmi="http://www.segula.fr/Anaxagore/CheckingValidation/Collection">
<DialogLinkList FunctionName="Transfert">
    <DialogLink id="1">
        <Task taskname="Démarrer Transfert"></Task>
        <Widget Category="PushButton"></Widget>
    </DialogLink>
    <DialogLink id="2">
        <Task taskname="Soute de départ"></Task>
        <Widget Category="drop-down list"></Widget>
        <UADialog ScriptLink="Fonctions\ElementsEntreeDepart\Transfert.txt"></UADialog>
        <SRDialog ScriptLink="ModelesDialogue_Lancement\Transfert\Transfert1.vbs"></SRDialog>
    </DialogLink>
    <DialogLink id="3">
        <Task taskname="Soute d'arrivée"></Task>
        <Widget Category="drop-down list"></Widget>
        <UADialog ScriptLink="Fonctions\ElementsEntreeArrivee\Transfert.txt"></UADialog>
        <SRDialog ScriptLink="ModelesDialogue_Lancement\Transfert\Transfert2.vbs"></SRDialog>
    </DialogLink>
    <DialogLink id="4">
        <Task taskname="Entrer le volume à transférer"></Task>
        <Widget Category="Edit"></Widget>
        <SRDialog ScriptLink="ModelesDialogue_Lancement\Transfert\Transfert3.vbs"></SRDialog>
    </DialogLink>
    <DialogLink id="5">
        <Task taskname="Valider"></Task>
        <Widget Category="PushButton"></Widget>
        <SRDialog ScriptLink="Generalisations\ScriptsGeneralises\SriptsLancement\Transfert.vbs"></SRDialog>
    </DialogLink>
    <DialogLink id="6">
        <Task taskname="Arrêter Transfert"></Task>
        <Widget Category="PushButton"></Widget>
        <SRDialog ScriptLink="Generalisations\ScriptsGeneralises\SriptsLancement\Transfert_ArretOperateur.vbs"></SRDialog>
    </DialogLink>
</DialogLinkList>
```

Figure 129 Structure de la collection de widgets suggérés

La Figure 130 présente le résultat des différents dialogues et de la présentation générée pour la fonction de transfert.

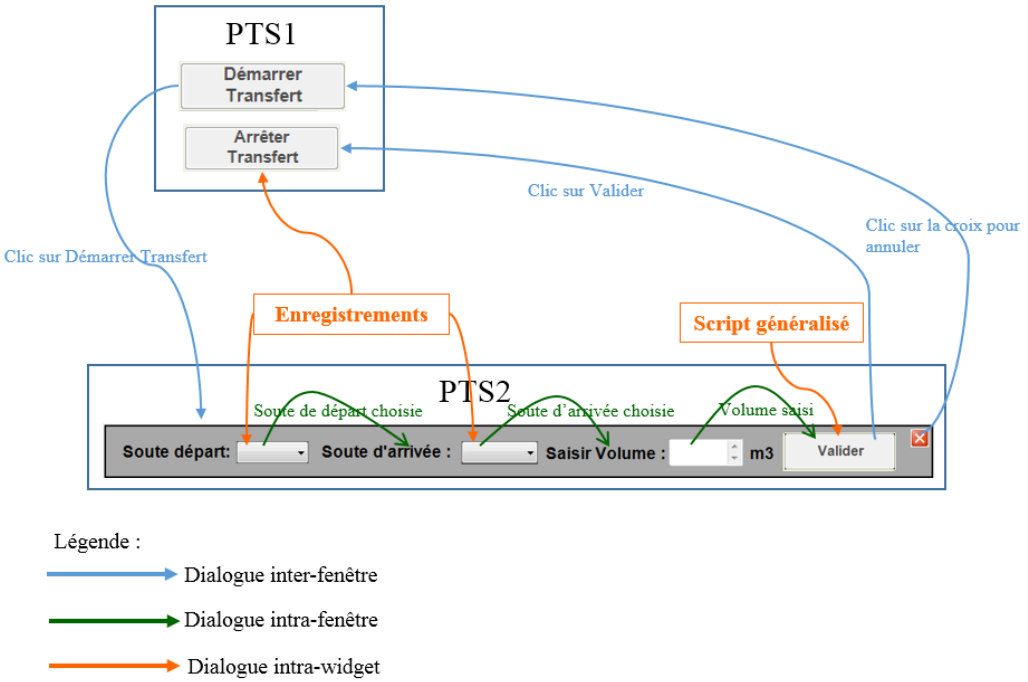


Figure 130. Synthèse de la génération du dialogue à tous les niveaux

L'application de la démarche illustrée sur la fonction de transfert à toutes les autres fonctions du système nous a permis de générer automatiquement le HLCCI (Figure 126).

4.3 Opération d'intégration d'interfaces de contrôle

L'exécution de la transformation d'intégration d'interfaces de contrôle renvoie une IHM d'ERGC dont le modèle EGRC contient l'interface de contrôle précédemment générée. Seule la partie fonctionnelle du modèle EGRC est impactée et prend en compte les interfaces générées pour les fonctions spécifiées.

La Figure 131 illustre le résultat de l'intégration de l'interface de contrôle de la fonction de transfert dans le modèle EGRC (représenté par le composant PUF sur Figure 131). On y retrouve les PTS ainsi que les différents dialogues enregistrés et/ou généralisés.

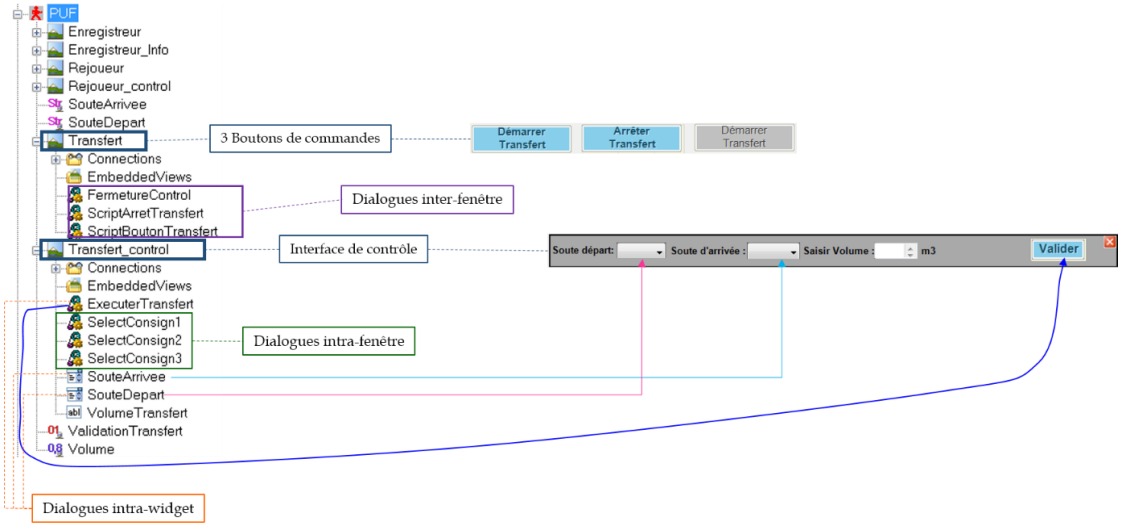


Figure 131 Résultat de l'opération d'Intégration de l'interface de contrôle de la fonction de transfert

4.4 Opération de test, débogage et paramétrage

L'opération de vérification et validation correspond à l'étape de rejeu, l'une des techniques d'EUD. Un script générique déjà existant dans le « Rejoueur » permet la lecture des différents dialogues générés pour les interfaces de contrôle, et l'affichage progressif des PTS permettant à l'utilisateur expert de vérifier et valider facilement les séquences et les configurations généralisés.

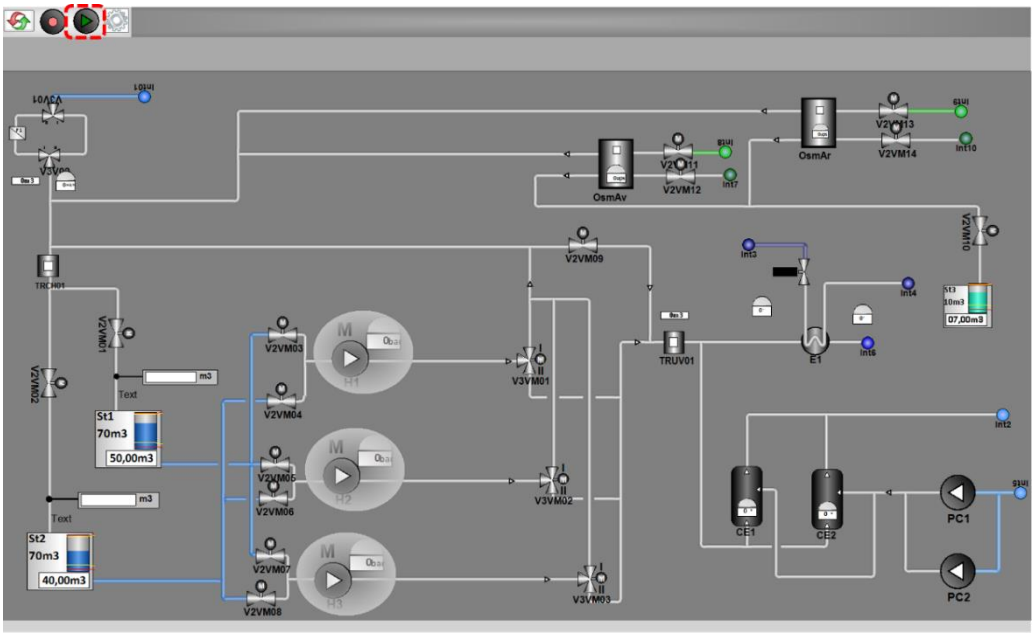


Figure 132 Rejoueur sur IHM de spécification

Pour vérifier et valider les spécifications fonctionnelles, l'utilisateur expert utilise le rejoueur (widget encadré en pointillé sur Figure 132). Lorsqu'il clique sur ce widget, le système lui affiche le HLCCI avec uniquement les widgets correspondants aux fonctions qu'il a déjà spécifiées. Pour l'illustration de la démarche, nous n'avons spécifié que la fonction de transfert (Figure 133).

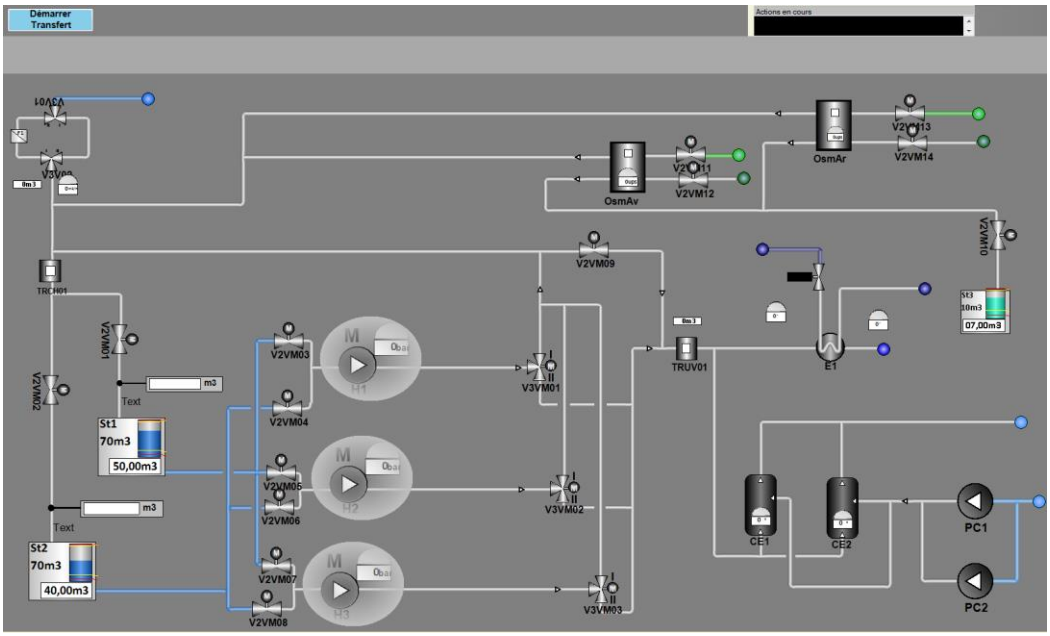


Figure 133 Comportement de l'IHM au cours de la phase de rejeu

L'utilisateur peut donc cliquer sur « *Démarrer Transfert* » (Figure 133) et suivre les différentes étapes proposées par l'interface pour rejouer les configurations généralisées de la fonction de transfert. Le système utilise, lors de ces étapes, les différents dialogues générés et les widgets suggérés pour guider l'utilisateur lors du rejeu. C'est aussi l'occasion pour l'utilisateur de vérifier les présentations des interfaces de fonctions, les dialogues et les configurations générés. Lorsqu'une configuration ne répond pas correctement à ses attentes, il peut la modifier en cliquant sur le bouton (Figure 134).

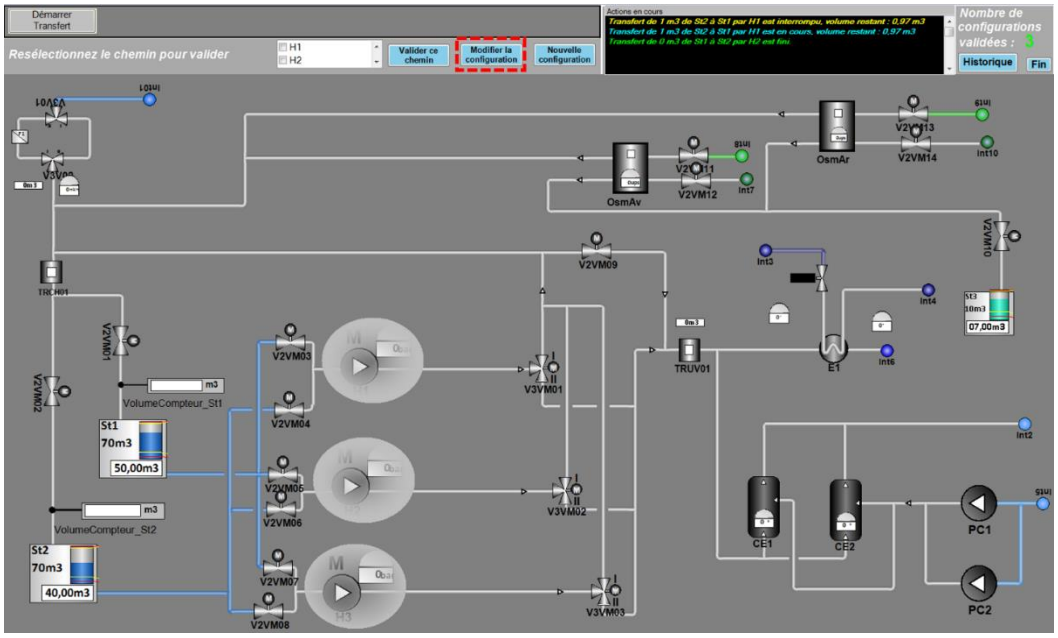


Figure 134 Modification d'une configuration

De la même manière, si les widgets suggérés lors de la spécification des exemples ne lui conviennent pas, il peut les modifier à travers le widget de paramétrage (encadré en pointillé rouge sur Figure 135).

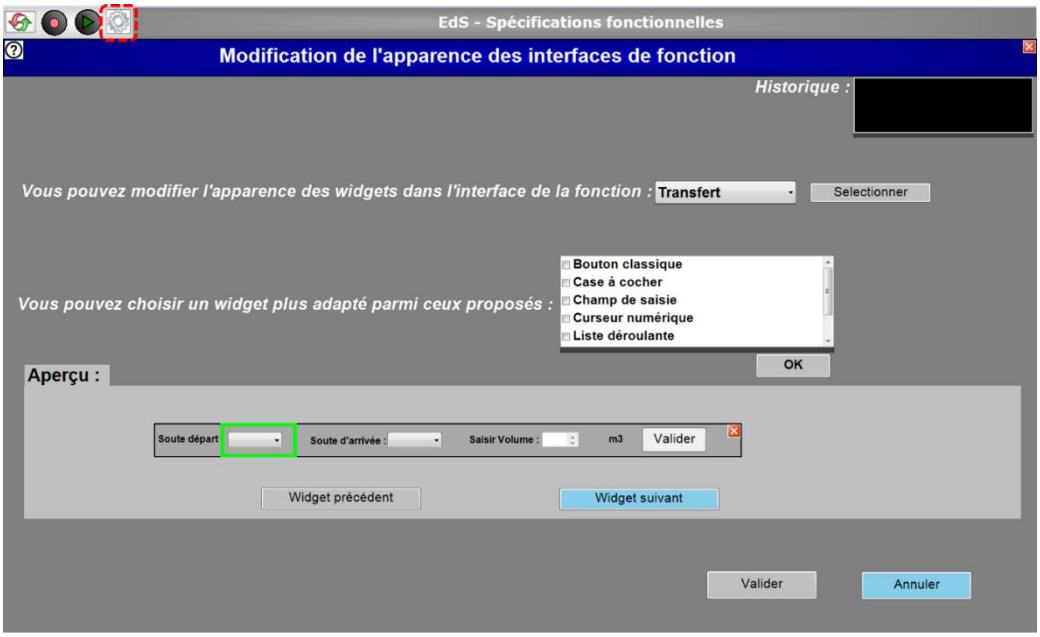


Figure 135 Modification de la présentation

Le clic sur ce widget donne accès à une interface qui permet de modifier les interfaces de fonction. Elle guide l'utilisateur tout au long du processus. Ce sont seulement les interfaces des fonctions spécifiées qui peuvent être modifiées. Quand l'utilisateur choisit une fonction dans la liste des fonctions qu'il a enregistrées, le système affiche l'interface correspondant à la fonction, accompagnée de boutons « Widget suivant » et « Widget précédent », qui permettent de passer d'un widget à un autre sur l'interface. L'utilisateur dispose d'un bouton « Valider » pour sauver ses modifications.

Les spécifications fonctionnelles (configurations, dialogues, présentations), ainsi validées avec la prise en compte des modifications faites par l'utilisateur, sont utilisées pour la génération de codes de commande de haut-niveau.

4.5 Opération de génération de codes de commande

L'exécution des transformations de génération de Grafcet, de génération de code ST et d'association renvoie le Grafcet de la Figure 136. La structure du grafcet est générée à partir des séquences d'actions faites par l'utilisateur lors de la spécification, tout en respectant l'ordre des actions. Les transitions liées aux Grafcets sont, quant à elles, générées à partir des configurations généralisées. En effet, on retrouve les ***configurations généralisées*** de la Figure 125 dans les transitions de la Figure 136.

En conclusion, l'ensemble des données (***programmes génériques, script enregistrés*** **et** ***configurations généralisées***) issues des spécifications fonctionnelles sont utilisées pour générer, d'une part, les interfaces de contrôle-commande globaux (HLCCI) et, d'autre part, les codes de commande pour les contrôles-commandes globaux qui communiquent avec les HLCCI à travers les variables de commande. Les contrôles-commandes globaux ainsi générés sont introduits dans l'IHM de spécification pour permettre la vérification et la validation des spécifications fonctionnelles.

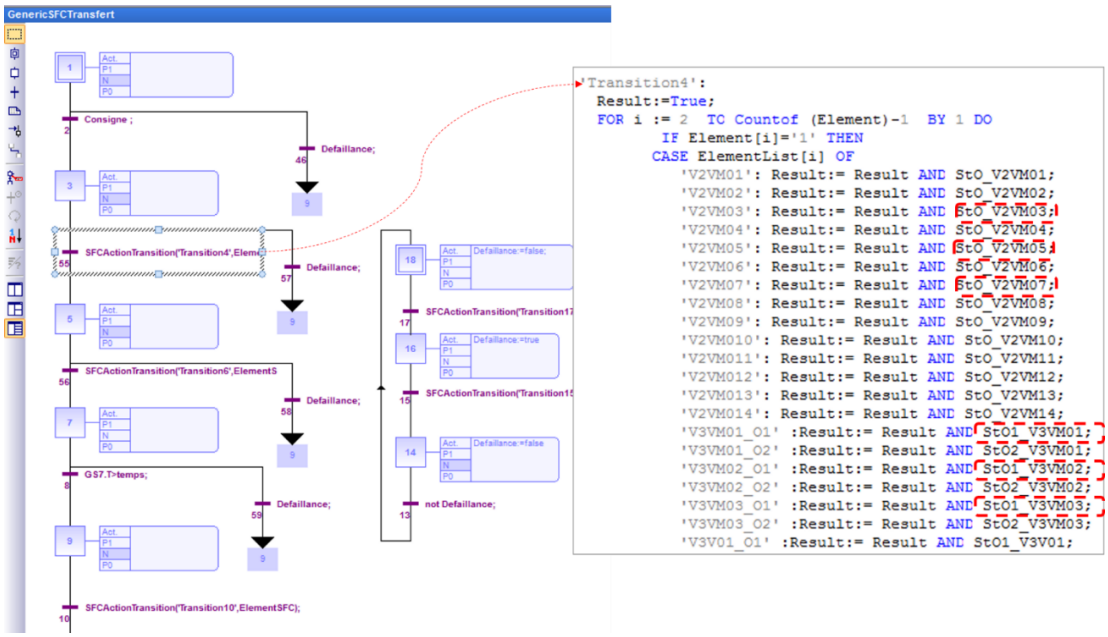


Figure 136 Extrait du résultat de la génération du code global de commande de la fonction de transfert

4.6 Opération de génération d'IHM

L'enrichissement de la structure et de l'architecture du projet d'IHM basique (Figure 137), issu des travaux de (Bignon 2012), renvoie l'IHM complète (Figure 138). Pour l'insertion des commandes globales, nous utilisons la zone de l'écran réservée aux boutons d'accès et aux dispositifs d'interface des commandes globales. Cette zone se situe dans la partie haute de l'écran, entre le bandeau d'alarme et la zone d'affichage principale (Figure 137).

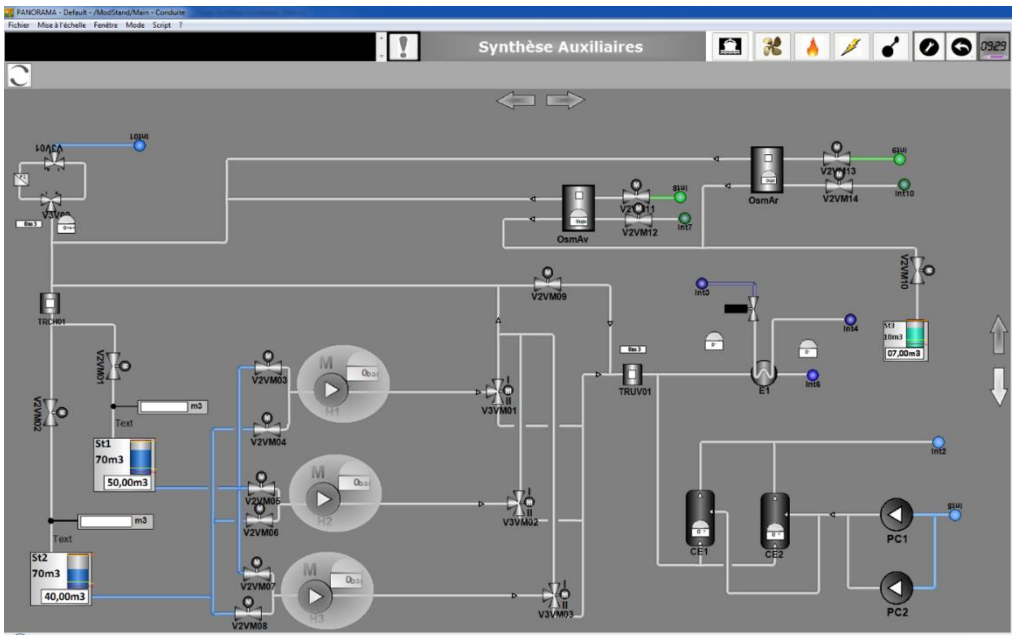


Figure 137 IHM basique – Résultat de l'opération d'insertion issue des travaux de (Bignon, 2012)

Ainsi, l'IHM complète contient l'IHM déjà générée mais on y trouve également les dispositifs d'interfaces permettant la saisie des consignes des commandes de haut-niveau (issues de l'opération de *vérification et validation*). L'IHM complète permet donc toujours une surveillance des éléments du système identifiés sur le synoptique, mais elle permet également de les contrôler de manière globale en déclenchant des séquences regroupant un ensemble de commandes élémentaires.

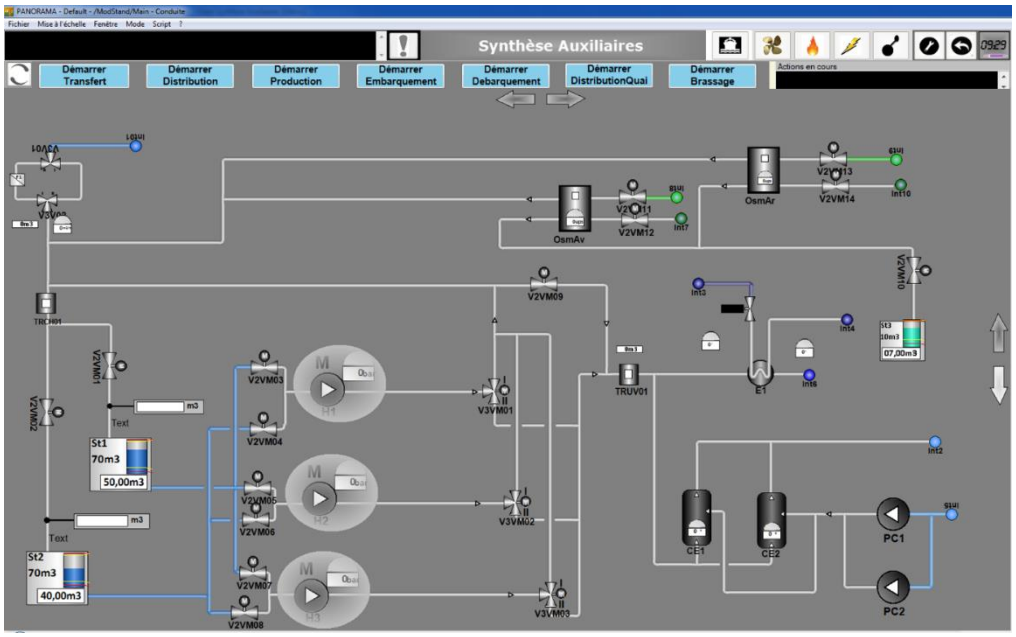


Figure 138 IHM complète - Résultat de l'opération de génération d'IHM

4.7 Opération de génération de commande

L'enrichissement du programme de commande vise à l'intégration des codes de commandes globales dans le programme élémentaire. Ces codes de commandes globales sont ajoutés au programme élémentaire sous la forme de blocs fonctionnels supplémentaires (codés en SFC) ; l'ensemble forme le programme complet.

Ainsi, le programme complet contient le programme élémentaire. Il contient également le code des commandes vérifiées et validées, qu'il met à disposition de la supervision au travers de variables de commande.

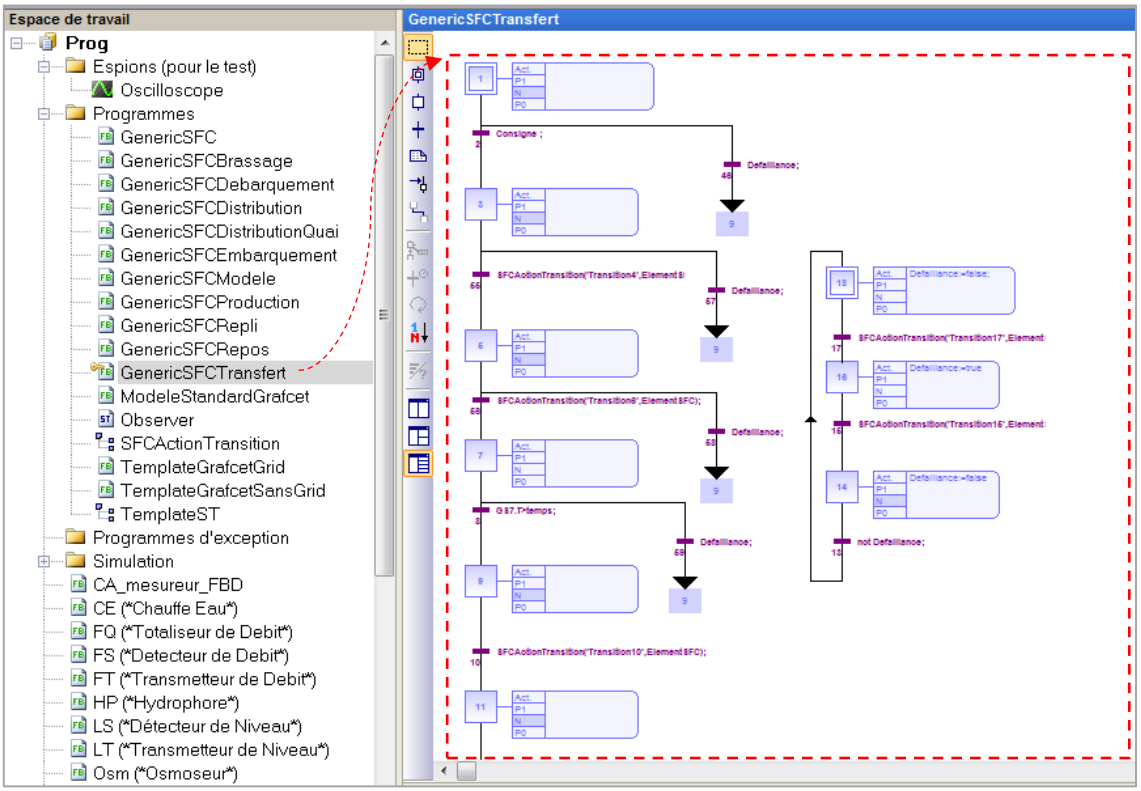


Figure 139 Résultat de l'opération de génération de commande

4.8 Conclusion

L'implémentation de notre démarche et son application à un exemple concret (le système EdS) nous a permis de valider l'ensemble de nos propositions facilitant la conception des systèmes de contrôle-commande. Le Tableau 5 donne une indication de la taille du système EdS en termes de nombre d'instances (éléments) et de connexions qu'il contient.

	Nombre d'unités
Instances	83
Connexions	126
Objets (instances + connexions)	209

Tableau 5 Composition du P&ID

Notre démarche commence par la conception des modèles de tâches. La proposition d'une interface (*Prototask Editor*) permet de mettre en œuvre le concept de patron pour faciliter l'obtention des modèles de tâches. La mise en œuvre de ce concept répond au besoin de standardisation souvent exprimé par les industriels. Les résultats obtenus par l'outil *Prototask Editor* sont encourageants, et l'intégration des modèles de tâches semble envisageable.

Une fois les modèles de tâches obtenus, nous avons cherché à obtenir les spécifications fonctionnelles en introduisant les techniques d'EUD dans une interface qui est familière aux concepteurs. La conception manuelle de cette interface est très fastidieuse car elle revient à implémenter 1296 objets catégorisés dans le Tableau 6. En utilisant la démarche de génération d'interface précédemment décrit dans (Bignon 2012), nous avons pu générer les parties de l'interface qui concerne les instances, les connexions, les équations et les équipotentiels, soit au total 346 objets automatiquement générés. Pourtant, cette démarche ne tient pas compte de l'introduction des techniques d'EUD. La prise en compte de cette technique implique la mise en œuvre de 950 objets, qui est aussi fastidieuse. En effet, 674 scripts permettent de lier les propriétés du modèle EGRC aux éléments du système et 276 scripts permettant de lier les propriétés des éléments du système au modèle EGRC, doivent être écrites manuellement.

Les objets de l'IHM d'EGRC	Nombre d'unités
Instances	69
Connexions	93
Équations de calcul de couleur de fluide	138
Équations de propagation d fluide	46
Liaison modèle EGRC - instances	**674**
Liaison instances – modèle EGRC	**276**
Total des objets	1296

Tableau 6 Composition de l'IHM d'EGRC

Nous avons donc, pour faciliter la conception, proposé une démarche qui englobe la génération à la fois des éléments (instances, connexions, équations d'animation et équipotentiels) de l'IHM et l'introduction des techniques d'EUD dans cette IHM. Notre démarche permet ainsi de générer automatiquement les **1296 objets** nécessaires pour le fonctionnement attendu de l'IHM d'EGRC.

La proposition d'une démarche de spécification fonctionnelle, basée sur combinaison de l'EUD et l'IDM a permis de réduire l'effort de spécification tout en obtenant des spécifications fonctionnelles vérifiées et validées à travers une interface d'EGRC. La vérification et la validation des spécifications fonctionnelles sont réalisées à partir des interfaces de contrôles

des commandes de haut-niveau, générées par la combinaison des modèles de tâches précédemment obtenus et des spécifications fonctionnelles généralisées.

Les spécifications fonctionnelles vérifiées et validées sont utilisées pour créer les codes de commande de haut-niveau présentés dans la section 4.5. Pour ce faire, un Grafcet et un code en langage St est défini pour chaque fonction nécessitant une commande de haut-niveau. L'implémentation manuelle de ces commandes de haut-niveau est longue et fastidieuse (plus le nombre de commandes de haut niveau est important, plus il faut du temps pour les implémenter manuellement). Par exemple, pour les 7 fonctions du système EdS, il faut implémenter 7 Grafcet et écrire 7 programmes en langage ST, puis relier ces programmes aux Grafcets. Nous avons donc proposé une démarche automatique permettant de générer ces Grafcets et codes associés en langage ST, puis les intégrer au programme de commandes élémentaires issus des travaux de (Bignon 2012), ceci afin de générer des gains de temps considérables et d'éviter d'éventuelles erreurs.

L'utilisation de l'IHM d'EGRC par l'expert non informaticien permet de résoudre les problèmes de communication et d'interprétation, ce qui peut réduire considérablement le temps de spécification fonctionnelle d'un système, donc de la réalisation du projet.

De plus, l'automatisation de l'obtention des interfaces de contrôle et des codes de commandes à partir des spécifications fonctionnelles garantit la **cohérence entre le programme de commande complet et l'IHM complète**.

Le Tableau 7 résume l'ensemble des opérations de notre flot et met en évidence les 5 phases manuelles qui nécessitent l'intervention de l'expert métier. Pour chacune de ces phases, notre démarche offre des outils à l'expert métier pour lui permettre d'exprimer facilement ses connaissances. Ces outils intègrent huit phases automatiques qui réduisent la charge de travail des concepteurs. Dès lors, le gain de temps est immédiatement visible.

Opération	Phase	Type
Adaptation des modèles de tâches	/	manuelle
Adaptation du modèle EGRC	/	automatique
Insertion Spec	/	automatique
Enregistrement-généralisation	Enregistrement	manuelle
	Généralisation	automatique
Génération d'interfaces de contrôle	/	automatique
Intégration d'interfaces de contrôle	/	automatique
Test, débogage et paramétrage	Test	manuelle
	Débogage	manuelle
	Paramétrage	manuelle
Génération de codes de commande	/	automatique
Génération d'IHM	/	automatique
Génération de commande	/	automatique

Tableau 7 Temps d'exécution de chaque opération sur la fonction de transfert

L'ensemble de notre démarche a été appliquée avec succès à la spécification des 7 fonctions du système EdS et à la génération du système de contrôle-commande complet.

Par ailleurs, dans le but de vérifier l'utilité de notre démarche de spécification fonctionnelle et l'utilisabilité de l'IHM d'EGRC, nous avons réalisé deux évaluations.

5 Évaluations

L'extension d'*Anaxagore* et son application à un système concret, nous a permis de valider notre approche d'utilisation des principes du EUD pour la description des spécifications des systèmes complexes. Pour intégrer ces principes dans Anaxagore, nous avons, d'une part, proposé une interface pour faciliter la spécification des tâches humaines et, d'autre part, généré une IHM de spécification pour faciliter la spécification fonctionnelle des systèmes complexes.

Ces deux systèmes interactifs proposés doivent répondre à des besoins fonctionnels et avoir les qualités requises pour être plus accessibles au monde industriel. Ces deux notions, issues du génie logiciel, sont l'utilité et l'utilisabilité, qui sont les piliers de l'évaluation d'un système en ergonomie (Senach 1990).

Dans le cadre plus précis de l'architecture globale d'un système interactif, l'utilité du système est vérifiée au niveau du noyau fonctionnel du logiciel tandis que l'utilisabilité est vérifiée au niveau des parties constitutives de l'IHM. En effet, l'utilité a trait à l'adéquation fonctionnelle : le logiciel permet-il à l'utilisateur d'atteindre ses objectifs de travail ? Et l'utilisabilité concerne l'adéquation de l'IHM : le logiciel est-il facile à apprendre et facile à utiliser ?

L'évaluation de ces deux notions conduit à la mise en place d'expérimentations. Dans la littérature, il existe deux types de méthodes d'évaluation : les méthodes prédictives et les méthodes expérimentales (Nielsen et Molich 1990a; C. Bastien 1991). Les méthodes prédictives (ou analytiques) ne nécessitent pas la présence de l'utilisateur final ni l'implémentation, même partielle, du système. De fait, elles conviennent aux toutes premières étapes du cycle de vie d'un logiciel : analyse des besoins et conception. Ces techniques permettent, à partir d'une description du système, voire de l'utilisateur, d'identifier des problèmes potentiels d'utilisabilité. À l'inverse des méthodes prédictives, les méthodes expérimentales requièrent la présence d'une population représentative du futur utilisateur du système et fournissent des données observées du monde réel.

Nous avons utilisé ces deux méthodes pour évaluer nos systèmes interactifs. Ces évaluations, ainsi que leurs résultats, sont présentés dans cette section.

5.1 Évaluation 1

5.1.1 Objectif et contexte

L'objectif de cette première évaluation est de vérifier d'une part les bénéfices de l'utilisation de l'EUD dans une démarche de spécification fonctionnelle et, d'autre part, d'évaluer l'utilisabilité de l'interface de spécification proposée. L'intérêt sera de vérifier à posteriori, les choix de conception de l'interface, d'impliquer l'utilisateur (concepteur ou expert mécanicien) dans la conception de l'interface de spécification et d'identifier les points à modifier afin de rendre cette dernière plus utilisable. Pour cela nous avons évalué la spécification fonctionnelle de la fonction *Transfert* auprès d'utilisateurs expérimentés, puis nous avons soumis l'interface à un audit ergonomique.

5.1.2 Tests utilisateurs

5.1.2.1 Méthode

La méthodologie employée pour ces expérimentations regroupe deux méthodes d'évaluation. La première est le test utilisateur, qui consiste à faire tester le système par un panel

d'utilisateurs représentatifs du public cible. Il permet d'observer et d'analyser le comportement des utilisateurs face au système. Ainsi, le test utilisateur permet de comprendre leurs objectifs réels et de relever les difficultés d'utilisation. C'est l'une des méthodes les plus efficaces pour améliorer l'ergonomie d'un système ou produit car elle permet de relever jusqu'à 95% des problèmes d'ergonomie (Boucher, 2009).

La deuxième est l'entretien semi-directif, qui consiste à recueillir des informations spécifiques et de nature qualitative (Caumont 2010). Cette technique est souvent employée pour la réalisation d'études exploratoires afin d'améliorer la connaissance d'un champ d'étude dont les thèmes essentiels sont familiers aux acteurs, mais qui présentent des aspects qui méritent un approfondissement.

Participants

Nous avons confronté l'interface à cinq participants (masculins) ayant entre 43 et 60 ans (49 ans en moyenne). Ils disposaient d'une expérience en navigation allant de 1 à 23 ans (9 ans et 1 mois en moyenne) et bénéficiaient tous d'une très bonne connaissance du système qui leur était présenté (ils en connaissaient le fonctionnement et l'utilité). En effet, ils avaient déjà navigué sur des navires marchands ou de transport de passagers de toutes tailles, et certains de ces navires disposaient d'un système similaire au système EdS. Les participants étaient donc familiarisés à ce type de système. Cependant, aucun d'entre eux n'a contribué à la conception de ce type de système.

Matériel

Le dispositif expérimental était composé d'un ordinateur relié à un écran de 23 pouces, d'une souris et d'un clavier.

Protocole

L'expérimentation s'est déroulée dans une salle réservée à cet effet, à l'École Navale Supérieure Maritime (ENSM) de Nantes. Les participants s'asseyaient face à l'IHM et l'expérimentateur à côté d'eux.

Pour évaluer leur compréhension du schéma P&ID du système EDS construit par le mécanicien, il leur était demandé de décrire oralement le schéma dans un premier temps (à quoi il sert, comment le système fonctionne…). Dans un second temps, il leur était demandé de montrer une configuration possible pour réaliser la fonction transfert à l'aide du même schéma. Cette première étape a été évaluée à l'aide d'une grille d'analyse spécialement conçue (Annexe 4). Nous avons découpé le schéma P&ID en plusieurs zones. Pour chaque zone, nous avons distingué la compréhension du fonctionnement des éléments vs. la compréhension de l'agencement du circuit (sens de circulation, entrées, sorties), que nous notions de 0 à 2 en fonction de la compréhension. Cette grille permet d'obtenir un score sur 20.

Ensuite, au cours d'une phase d'entretien, nous avons cherché à vérifier si la méthode utilisée pour définir (spécifier) la fonction de transfert, puis pour la lancer (en utilisation) était correcte. Pour cela, une simulation de modèles de tâches leur était présentée à travers l'outil Prototask (les deux modèles de tâches décrivent comment réaliser le transfert avec et sans commande de haut niveau).

Les participants étaient ensuite formés à l'utilisation de l'interface de spécification (la partie de l'interface qu'ils ne connaissaient pas) à travers une démonstration pendant environ 15 min, puis il leur était demandé de l'utiliser pour spécifier la fonction *Transfert*. Enfin, une phase

d'entretien avec les participants nous permettait de récolter leurs impressions et leurs critiques pour améliorer l'outil. Le guide d'entretien utilisé pour notre expérimentation était articulé autour de douze questions, se rapportant à trois thèmes principaux. Ils ont été définis, d'une part, par rapport à la connaissance que nous avions et voulions acquérir du sujet et d'autre part, par rapport à l'utilisation et l'amélioration de l'outil. Il leur était notamment demandé d'évaluer le niveau de difficulté des actions effectuées sur l'IHM sur une échelle allant de 1 « très facile » à 10 « très compliqué ». À des fins d'analyse, l'expérimentation a été enregistrée puis retranscrite.

5.1.2.2 Analyse des résultats

Évaluation de la compréhension du PID

Tous les participants ont fait une bonne description du système (Figure 140). Ils connaissaient l'architecture du système et son utilisation, et connaissaient chacun des éléments et leurs rôles dans le système. Les scores de description du PID sont de 19.40 en moyenne (ET = 0.89).

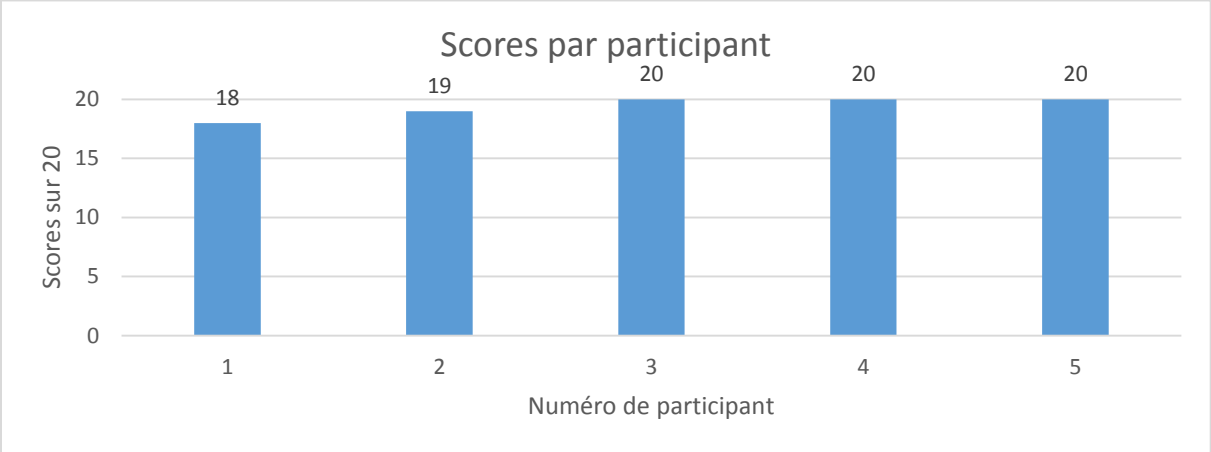


Figure 140 Évaluation de la compréhension du PID grâce à la grille d'évaluation

Cette étape nous a permis de confirmer que même si les participants n'ont pas un profil « concepteur », ils ont suffisamment de connaissance du système pour en décrire les spécifications.

Modèles de tâches

Modèle de tâche séquentiel

En ce qui concerne les tâches décrites pour la réalisation d'un transfert manuel, les participants projettent la réalisation de la fonction sur d'autres systèmes. De l'analyse de cette partie résultent deux scénarios possibles :

Scénario 1 : Transfert par pompe sans sécurisation. Pour ce type de transfert, on choisit le circuit, puis on ouvre les vannes concernées et on démarre la ou les pompes.

Scénario 2 : Transfert par pompe avec sécurisation. Dans ce cas, on choisit la ou les pompes à utiliser, on ouvre les vannes autour dans le respect des règles de conduite. Ensuite, on ouvre la ou les pompes choisies, puis on sécurise le circuit en vérifiant que les éléments qui peuvent altérer la fonction sont fermés.

Nous avons constaté que le scénario 1 est le plus utilisé des deux par les participants. Cette analyse nous a permis de remarquer que les séquences d'actions élémentaires pour la réalisation d'une même fonction peuvent varier en fonction du type de système. Ainsi, nous

avons validé l'utilité de notre démarche qui est de capturer pour chaque système, les connaissances de l'expert afin d'avoir les bonnes spécifications.

Modèle de tâches abstrait

Les tâches décrites pour la réalisation d'un transfert en utilisant une commande de haut-niveau ont toutes été validées à l'unanimité. Nous avons pu ainsi vérifier la conformité du modèle de tâches de fonctions ayant servi à la conception de l'interface de spécification. Sur les systèmes qu'ils ont connus toutes les fonctions se faisaient manuellement ; c'est-à-dire pour réaliser une fonction, les opérateurs commande (ouvre ou ferme) les éléments concernés dans un ordre bien précis qu'ils connaissent. Cependant, l'idée de définir une commande globale pour le lancement d'une fonction a été très bien acceptée avec une suggestion de ne pas pour autant supprimer le mode manuel. L'opérateur doit avoir le choix de lancer la fonction de façon semi-automatique ou manuelle. Ils ont tous été rassurés à cet effet.

Test de l'interface

L'analyse des interactions des participants avec l'interface nous a permis de dégager un certain nombre de problèmes.

La sélection des départs et arrivées

Nous avons constaté que les participants étaient un peu perdus lors de l'enregistrement des arrivées. En effet, les bandes de départ et arrivées s'affichent en même temps lors de l'enregistrement de cette phase (Figure 141). Les participants se demandaient où cliquer et pourquoi l'étape d'enregistrement des départs était toujours présente à l'écran.

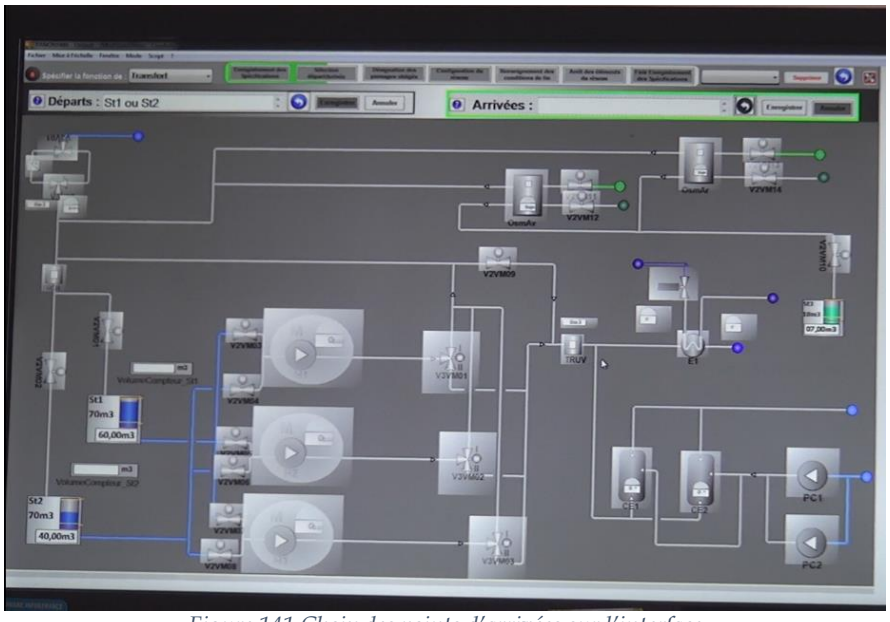


Figure 141 Choix des points d'arrivées sur l'interface

Les points de passage obligés

La notion de points de passages possibles ne semble pas évidente pour les participants. Ils ont néanmoins bien compris ce qu'il fallait faire grâce à la phase de formation à l'IHM. Cependant, ils n'ont pas de proposition à nous faire pour améliorer ce point.

La configuration du réseau

Les participants ont été un peu déroutés par les termes « configuration du réseau ». En effet, arrivés à cette étape, ils ouvraient les vannes du circuit mais ne pensaient pas à démarrer les pompes. La « configuration du réseau » serait plutôt une « disposition du circuit », mais cette dernière ne tient pas compte de la disposition des pompes, elle concerne uniquement l'ouverture des vannes.

Le renseignement des conditions d'arrêt

Le renseignement des conditions d'arrêt devait s'effectuer grâce à un « glisser/déposer » qui a fortement perturbé les participants (Figure 142). En effet, ils ne comprenaient pas comment renseigner la condition d'arrêt. Ce mode d'interaction n'est pas du tout intuitif pour eux.

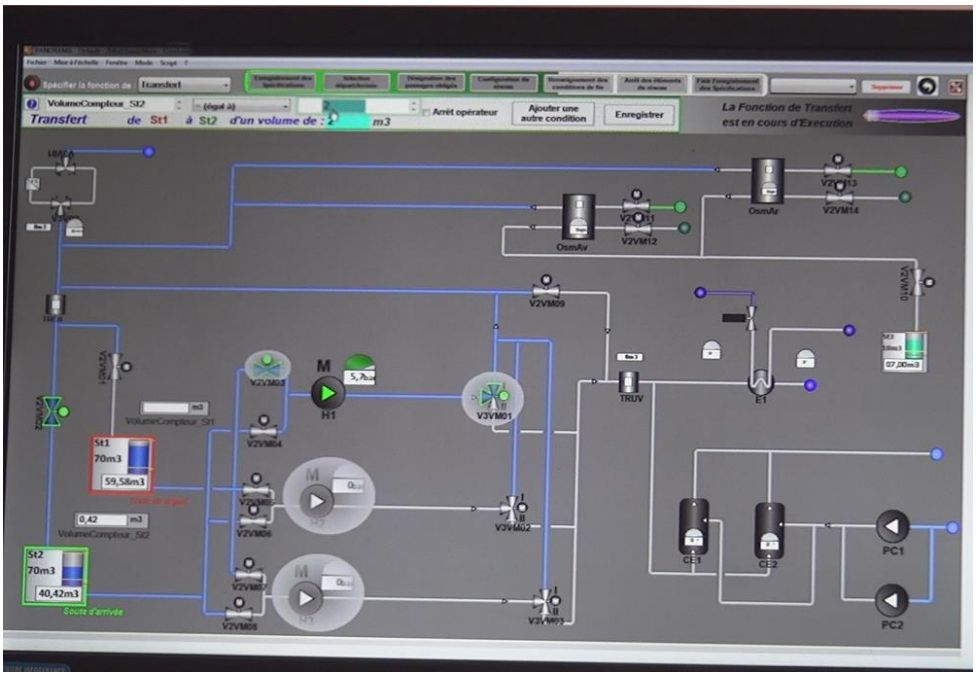


Figure 142 Glisser/Déposer

Les autres étapes de la phase d'enregistrement se sont bien passées. Le rejeu aussi a été fait sans erreur par les participants.

Entretien post-test

Concernant la mise en œuvre de la spécification fonctionnelle à travers l'interface proposée, les participants ont relevé le manque de spécification de la « sécurisation » du système. Selon les participants, certaines erreurs de spécification apparaissent dans la définition de toutes les sécurités liées à chacune des fonctions, pour la robustesse du système. Il est donc essentiel de compléter les phases de spécification par une définition de contraintes permettant de rendre le système sûr suivant les différentes configurations.

D'un point de vue ergonomique, le manque d'information au niveau de la phase de rejeu a été remarqué par les participants. Pendant cette phase, le concepteur doit lancer les fonctions afin de vérifier et de valider les spécifications produites par le système. Cependant, à partir du moment où il lance une fonction, il n'a aucune information sur cette dernière.

Les participants trouvent important que l'utilisateur puisse à tout moment connaître les fonctions en cours sur son système. Pour le passage d'une étape de spécification à une autre, il faut que les participants cliquent sur le bouton correspondant à l'étape à réaliser mais ceci

n'est pas du tout évident pour eux malgré les messages pour les guider. Les différentes étapes proposées sur l'interface pour atteindre la spécification d'une fonction ont toutes été validées.

Sur une échelle de 1 à 10, avec 1 « très facile » et 10 « très compliqué », le niveau de difficulté d'utilisation de l'IHM de spécification est de 4 en moyenne. Deux participants ont trouvé le démonstrateur facile d'utilisation (note de facilité d'utilisation inférieure à 3/10), trois participants ont éprouvé davantage de difficulté (notes comprises entre 5 et 6/10). Les participants ont tous signalé le fait que l'utilisation devient beaucoup plus facile après la spécification de la première fonction.

5.1.3 Audit ergonomique

Pour compléter cette analyse, un audit ergonomique de l'IHM basé sur les dix principes heuristiques de Jacob Nielsen (Nielsen et Molich 1990a), et sur les critères de Bastien et Scapin (C. J. M. Bastien et Scapin 1993) a été réalisé.

5.1.3.1 Méthode

L'audit ergonomique est une méthode, dite « experte », réalisée par un ergonome, qui consiste à analyser tout ou partie d'une interface en s'appuyant sur ses connaissances, expériences et convictions (Boucher 2009). Il fait appel à la capacité d'analyse de l'ergonome, ainsi qu'à son expérience et est soutenu par des critères ergonomiques (C. J. M. Bastien et Scapin 1993). Cette méthode est complémentaire à celle du test utilisateur.

5.1.3.2 Résultats

Le problème majeur mis en lumière par l'audit ergonomique de l'interface est un guidage inapproprié. En effet, le système utilise des formes de guidage explicites pour orienter et aider l'opérateur dans la réalisation de ses tâches. Ces formes de guidage nécessitent des actions superflues de la part de l'opérateur et augmentent son temps de réalisation des tâches.

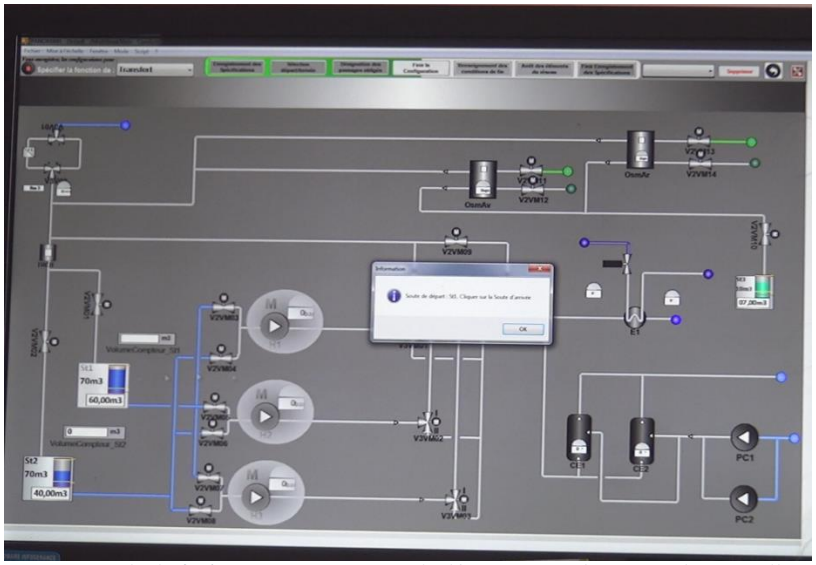


Figure 143 Exemple de fenêtre pop-up : « Soute de départ : St1. Cliquer sur la soute d'arrivée ».

Les fenêtres contextuelles (ou fenêtres pop-up) sont utilisées tout au long du processus de spécification fonctionnelle pour communiquer des instructions au concepteur et le guider (Figure 143). Cette forme de guidage explicite n'est pas appropriée car elle demande du temps et de l'attention supplémentaire à l'opérateur qui doit prendre connaissance de ces messages puis effectuer les actions nécessaires à la tâche. Les fenêtres augmentent considérablement le

nombre de clics que l'opérateur doit effectuer au cours du processus ainsi que sa charge mentale. De plus, il y a un risque que l'opérateur valide le message sans l'avoir lu, et soit donc perdu sur l'interface. En effet, les fenêtres contextuelles sont intrusives car elles interrompent brutalement l'opérateur dans sa tâche et peuvent être perçues comme indésirables. Il faut réserver ce mode de communication aux informations de haute importance pour la tâche et pour lesquelles il est nécessaire de s'adresser directement à l'opérateur.

Les boutons de confirmation de début et de fin de tâche nécessitent des actions qui devraient être prises en charge par le système (Figure 144). Là encore, on utilise une forme de guidage explicite qui nécessite des clics superflus. En effet, le passage d'une étape à l'autre devrait se faire via le bouton « valider » ou « enregistrer » qui est déjà présent et sur lequel l'opérateur doit cliquer à chaque fois. De plus, ces boutons sont mal placés car ils se situent dans la barre de progression et remplacent temporairement le nom de l'étape en cours.

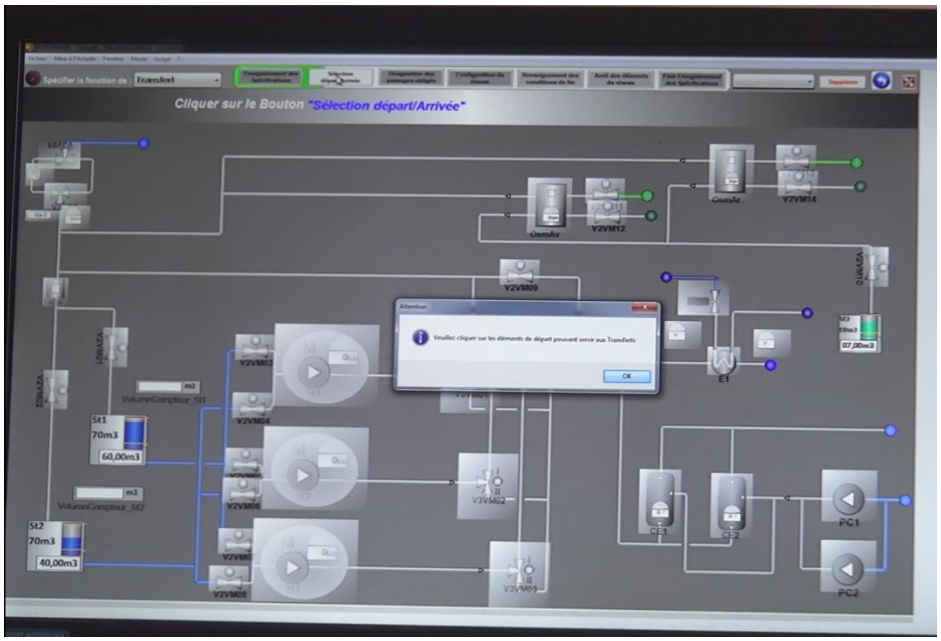


Figure 144 Exemple de bouton de confirmation de début de tâche : « Sélection départ/arrivée »

L'emplacement et l'apparence des éléments sont sujets à confusion. En effet, le système n'indique pas toujours clairement quelles sont les actions à effectuer pour réaliser la tâche, comment les effectuer et dans quel ordre. À certaines étapes du processus par exemple, l'opérateur doit effectuer une séquence d'actions dans un ordre précis, alors que l'interface lui fournit l'ensemble des champs à remplir en même temps, ce qui peut entraîner des difficultés. L'utilisation du glisser/déposer constitue un autre problème de guidage car aucun indice ne laisse penser à l'opérateur que ce type d'interaction est possible avec le système. De plus, il n'existe pas de tutoriel pour guider et informer l'opérateur lors de la première utilisation.

L'ensemble de ces éléments rend le processus très long à effectuer, très coûteux d'un point de vue cognitif, et déplaisant pour l'opérateur. Les conclusions de l'audit invitent à repenser l'interaction homme-machine en passant d'un guidage explicite à un guidage implicite en agissant sur l'agencement des champs à l'écran, leur ordre d'apparition ainsi que leurs représentations graphiques pour rendre le processus plus rapide, intuitif et autonome.

5.1.4 Discussion

Des résultats différents mais complémentaires découlent de ces deux évaluations.

Tout d'abord, les tests utilisateurs ont permis de démontrer que l'approche permet de faciliter la spécification fonctionnelle des systèmes complexes. Elle a été bien reçue par les participants, ce qui démontre son acceptabilité. Tous les participants ont apprécié la phase de rejeu pendant laquelle le système leur propose une maquette interactive de la commande de haut niveau qu'ils spécifient, telle qu'elle sera présentée sur l'IHM finale, et de vérifier et de valider les configurations générées par l'outil.

Outre les problèmes d'interaction qui alourdissent le processus, d'importantes informations sur le processus de spécification permettront d'améliorer les fonctionnalités du démonstrateur d'outil de spécification. En effet, pendant le processus de spécification, tous les participants ont relevé le nombre important de clics à faire sur l'interface pour la réalisation de la tâche de spécification.

L'audit ergonomique a, quant à lui, relevé des problèmes liés à la *longueur du processus*, *au guidage explicite gênant* et au *manque de tutoriel*. L'un de ces problèmes (la longueur du processus) ressort également de l'analyse des tests utilisateurs. Cependant, des points positifs ont également été relevés. Ces points sont jugés en s'appuyant sur les critères définis par les travaux de (Nielsen et Molich 1990b; C. J. M. Bastien et Scapin 1993). En effet, la barre d'outils de l'interface est composée à 100% d'icônes en accord avec la situation de référence, ce qui facilite la compréhension de la tâche de l'utilisateur. Pendant le processus de spécification, l'utilisateur a toujours la main pour contrôler les traitements en cours ce qui permet d'éviter des erreurs. Le système permet d'annuler et de refaire une action et fournit un retour approprié, en temps réel, à l'utilisateur.

L'interface dispose également d'une icône d'aide qui permet de fournir une aide à l'utilisateur s'il est perdu. En effet, bien qu'il soit préférable que le système puisse être utilisé sans le recours à une documentation, il peut cependant être nécessaire de fournir de l'aide et de la documentation. Les informations de ce type devraient être faciles à trouver, centrées sur la tâche de l'utilisateur, indiquer concrètement les étapes à suivre et ne pas être trop longues (Nielsen et Molich 1990b).

L'une des préconisations de l'audit ergonomique est de raccourcir le processus de spécification fonctionnelle en retravaillant le guidage au sein de l'interface. Les propositions d'amélioration qui ont été présentées après cette expertise répondent à cet objectif puisqu'elles permettent de réduire de plus de 40% le nombre d'actions que l'opérateur doit effectuer. Elles permettent également de diminuer le nombre d'erreurs et d'augmenter l'utilisabilité perçue de l'interface au cours des prochaines sessions de test utilisateurs. Les informations issues de ces analyses sont utilisées pour rendre l'IHM plus intuitive à l'utilisateur. L'IHM est ensuite soumise à des tests utilisateurs plus stricts.

5.2 Évaluation 2

5.2.1 Objectif et contexte

L'objectif de cette deuxième évaluation est d'une part d'impliquer les futurs concepteurs dans la conception de l'interface de spécification, pour vérifier qu'ils auraient envie d'utiliser notre démarche de spécification. D'autre part, nous voulons nous assurer que l'interface correspondra à leurs besoins pour la spécification d'un système. Ainsi, nous vérifions à la fois l'utilité de notre démarche et aussi les apports des améliorations des précédents tests sur son utilisabilité. Pour cela nous avons évalué la spécification fonctionnelle sur trois fonctions : *Transfert*, *Débarquement* et *Distribution*, auprès de futurs concepteurs. L'intérêt dans le fait de

tester trois fonctions est de vérifier la facilité d'utilisation et de prise en main de l'interface par les futurs concepteurs.

5.2.2 Méthode

Participants

Pour cette seconde évaluation, nous avons de nouveau mis en œuvre la méthode du test utilisateur auprès de 16 étudiants en génie des procédés à l'école Polytech Nantes (9 hommes et 7 femmes). Ils étaient âgés de 21 à 25 ans (22,6 ans en moyenne) et bénéficiaient d'une connaissance variable des schémas P&ID.

Matériel

Le dispositif expérimental était composé d'un ordinateur relié à un écran de 23 pouces, d'une souris et d'un clavier, ainsi que d'un mémo récapitulant les grandes étapes de la spécification (Annexe 5). L'écran des participants était filmé durant les tests. Les participants devaient préalablement signer une demande d'autorisation d'enregistrement.

Protocole

À leur arrivée, un diaporama de présentation commenté a été présenté collectivement aux participants afin de replacer les expérimentations dans leur contexte. Les tests utilisateur se sont ensuite déroulés de façon individuelle dans des salles séparées (une salle par participant et par expérimentateur) dans les locaux de l'école. Les participants s'asseyaient face à l'IHM et l'expérimentateur à côté d'eux. Il y avait 4 expérimentateurs différents.

Pour évaluer leur compréhension du schéma P&ID du système EDS construit par le mécanicien, il leur était demandé aux participants dans un premier temps de décrire oralement le schéma (à quoi il sert, comment le système fonctionne…). Dans un second temps, il leur était demandé de montrer une configuration possible pour réaliser chacune des trois fonctions testées (Transfert, Débarquement et Distribution) à l'aide du même schéma. Dans le cas où les participants n'arrivaient pas à comprendre le schéma, les expérimentateurs leurs apportaient toutes les informations nécessaires à leur compréhension du schéma. Cette première étape a été évaluée à l'aide d'une grille d'analyse spécialement conçue (Annexe 4). Nous avons découpé le schéma PID en plusieurs zones. Pour chaque zone, nous avons distingué la compréhension du fonctionnement des éléments vs. la compréhension de l'agencement du circuit (sens de circulation, entrées, sorties), que nous notions de 0 à 2 en fonction de la compréhension. Cette grille permet d'obtenir un score sur 20.

Après cela, un second diaporama commenté était diffusé. L'objectif de ce diaporama était de présenter un tutoriel permettant d'appréhender un peu le fonctionnement de l'interface. Toutefois, nous avons veillé à ne pas donner de visuel de l'interface ou d'indications sur la façon dont les actions devaient être réalisées sur l'interface afin de préserver l'effet de découverte lorsqu'ils l'utiliseraient pour la première fois.

Ensuite, les participants devaient réaliser la spécification des fonctions Transfert, puis Débarquement, puis Distribution. Ils devaient tester le rejeu pour la première fonction au moins. Ils étaient invités à penser à voix haute pour mieux comprendre leurs actions sur l'interface.

Enfin, ils étaient invités à compléter le questionnaire SUS (System Usability Scale (Brooke 1996)) ainsi qu'un questionnaire plus ouvert leurs permettant de donner leurs impressions sur l'interface (Annexe 6).

5.2.3 Résultats

5.2.3.1 Évaluation de la compréhension du PID

La moyenne des scores est de 9,75 (ET = 7,66 ; Min = 0 ; Max = 20), ce qui confirme que les participants n'étaient pas tous familiers avec ce type de schéma. Lorsque l'on observe les différents scores (Figure 145), on constate que la compréhension du schéma peut varier du tout au tout selon le participant.

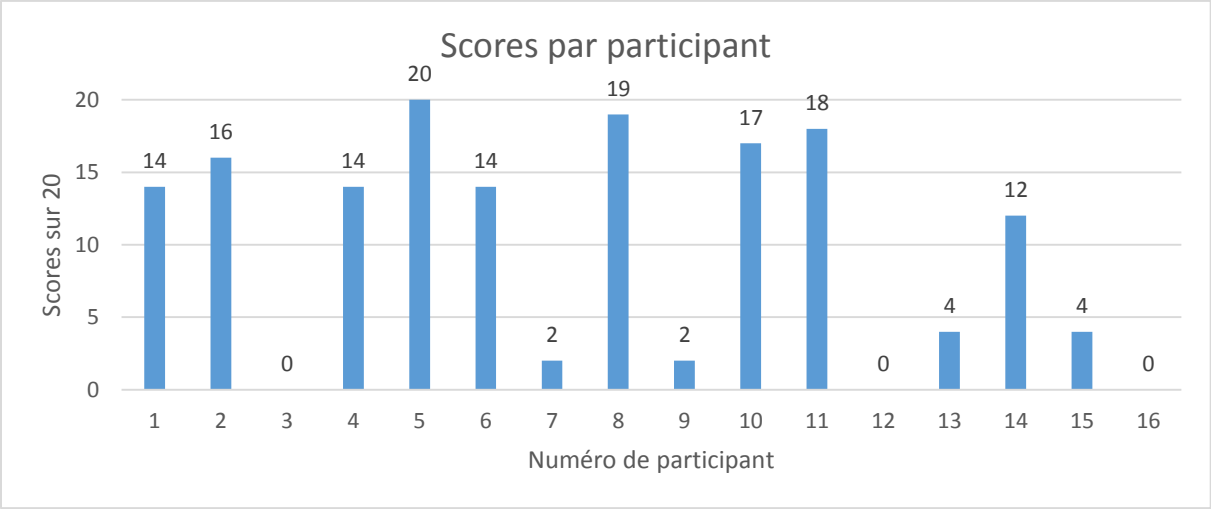


Figure 145 Scores par participant (sur 20 points)

Cette hétérogénéité des résultats peut être expliquée par la diversité des profils des étudiants qui n'ont pas tous suivi le même cursus. Les difficultés des participants à interpréter le schéma ont été compensées par des explications des expérimentateurs afin de leur permettre d'avoir les informations nécessaires pour le test.

5.2.3.2 Évaluation de l'utilisabilité de l'interface

5.2.3.2.1 Temps moyens

Sur le graphique ci-dessous (Figure 146), on peut constater que le temps mis pour réaliser la spécification des fonctions (rejeu compris) diminue nettement avec l'expérience. Cela témoigne d'un apprentissage du fonctionnement de l'interface. C'est encourageant car cela montre qu'après une prise en main, l'outil est utilisé plus facilement.

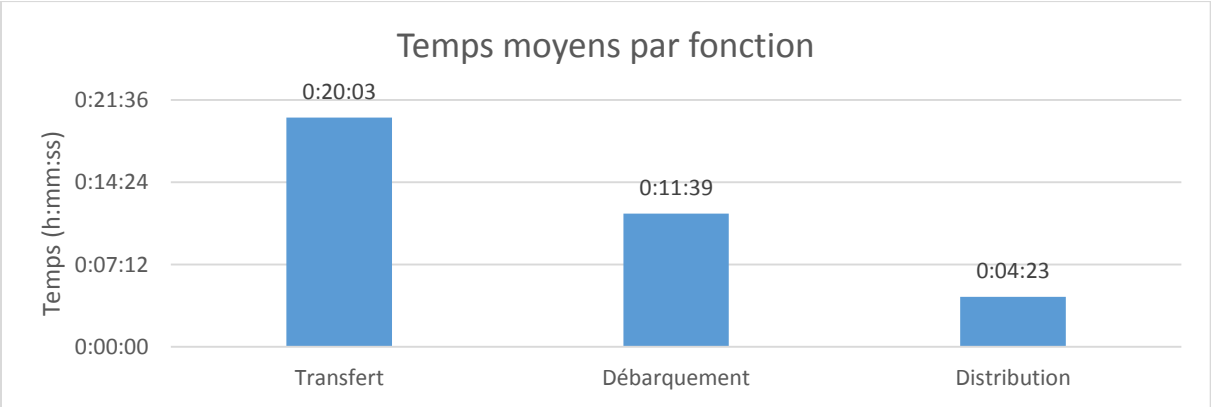


Figure 146 Temps moyens de réalisation des fonctions

On retrouve ce même effet d'apprentissage lorsqu'on observe le temps de réalisation de chaque étape (Figure 147). À première vue, l'étape qui a demandé le plus de temps aux participants est la configuration du circuit.

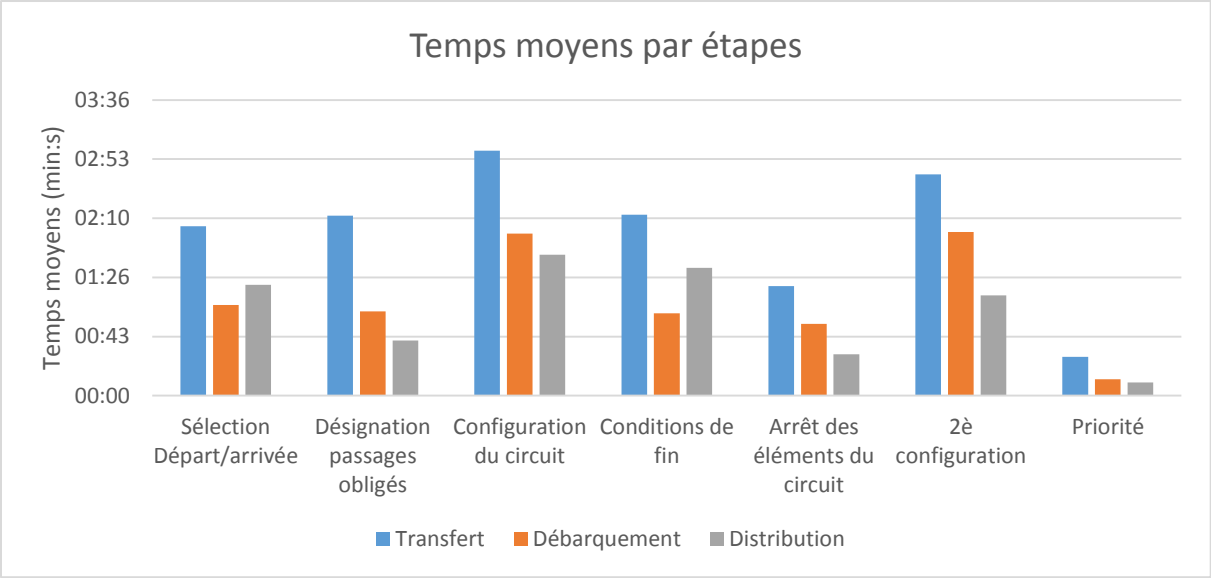


Figure 147 Temps moyens de réalisation des étapes selon la fonction

5.2.3.2.2 Ratio succès/échecs

Grâce au graphique ci-dessous (Figure 148), nous pouvons constater que le rejeu est l'étape qui a posé le plus de problèmes. En effet, c'est celle qui comporte le plus grand nombre d'échecs, toutes fonctions confondues. En ce qui concerne la partie enregistrement, les fonctions transfert et débarquement sont relativement équivalentes en terme de succès et de redémarrages nécessaires.

L'analyse des résultats montre ici qu'il reste encore des problèmes d'utilisabilité de l'interface à corriger. En effet, le nombre de redémarrages et d'échecs au rejeu traduisent de réelles difficultés qui doivent impérativement être corrigées.

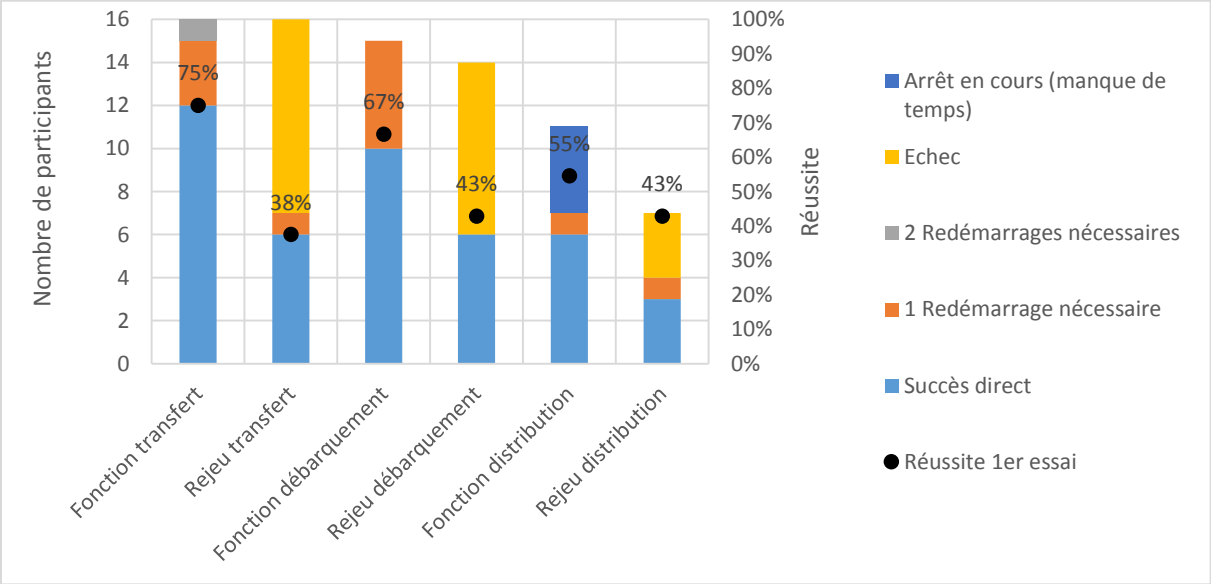


Figure 148 Ratio succès/échec selon les fonctions

5.2.3.2.3 Problèmes relevés

Dans cette section, nous allons dresser une liste des problèmes relevés au cours des tests. Cette liste est non exhaustive et ne rapporte que les problèmes « majeurs ». On trouvera en annexe un tableau plus complet comportant des extraits des tests regroupés par type de problème.

Bugs d'interface

100% des participants ont rencontré au moins un bug dit « bloquant », c'est-à-dire qui empêche toute progression, au cours des tests. 15 participants ont rencontré un bug ou plus lors de la phase d'enregistrement, et 14 participants lors de la phase de rejeu. Chaque participant a rencontré en moyenne 3,5 bugs au cours des tests (ET = 1,71 ; Min = 1 ; Max = 7), ce qui porte le nombre total de bugs (tous participants confondus) à 56.

Ces chiffres conséquents indiquent que le système doit encore être perfectionné afin de se prémunir des éventuelles erreurs des utilisateurs. De même, sa stabilité doit pouvoir être garantie d'un poste à un autre.

Les commandes des vannes motorisées et des pompes

14 participants sur 16 ont rencontré des difficultés avec les commandes des vannes et des pompes au cours des tests (Figure 149), soit 87,5% des participants. Pour 4 d'entre eux, le problème a été rencontré plusieurs fois, soit 28.57%.

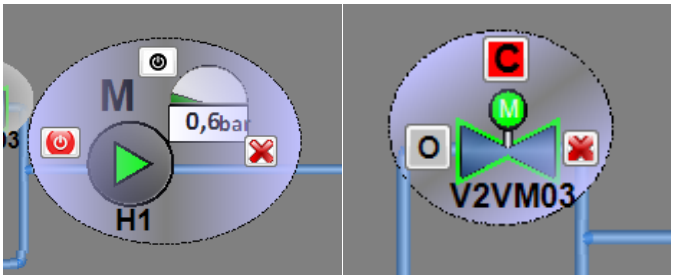


Figure 149 Commandes des pompes (à gauche) et des vannes (à droite)

Bien souvent, il y a confusion entre le bouton en forme de croix rouge qui permet de fermer le volet de commande de l'élément et le bouton qui sert à arrêter/fermer l'élément, de couleur rouge lui aussi. Les commentaires ci-dessous illustrent les difficultés rencontrées par les participants :

« *Bah en fait j'hésitais parce que je me dis, si j'appuie sur la croix, est-ce que ça va me fermer la vanne. En fait. Ou l'ouvrir, enfin… Après c'est vrai que de base tout est fermé mais bon là je me dis, est-ce qu'en cliquant dessus je ne l'ai pas ouverte et si j'appuie sur la croix ça va me la fermer.* » (Participant 1).

« *Par exemple (en montrant l'option fermer le cercle des options de la vanne) j'aurais pensé que c'était pour fermer la vanne.* » (Participant 9).

« *(Clic sur la vanne après H1) Je ne vois pas trop la différence entre 'fermer la vanne' et 'fermer le menu circulaire'.* » (Participant 15).

Premièrement, la disposition des boutons à distance équivalente les uns des autres ne permet pas de distinguer ceux qui commandent l'élément de celui qui ferme le volet de commande. Deuxièmement, la couleur rouge et la forme des symboles ne permettent pas de les distinguer facilement. Les utilisateurs doivent alors fournir des efforts cognitifs importants pour déterminer sur quel bouton ils doivent cliquer. Ils doivent alors apprendre la fonction de chaque bouton au cours de leur utilisation de l'outil. C'est ce que tendent à montrer nos résultats puisque pour 13 d'entre eux, le problème a été rencontré uniquement durant la fonction transfert, soit 92,85 %.

Le bouton étape suivante

13 participants sur 16 ont rencontré des difficultés avec le bouton étape suivante, soit 81.25% des participants (Figure 150).

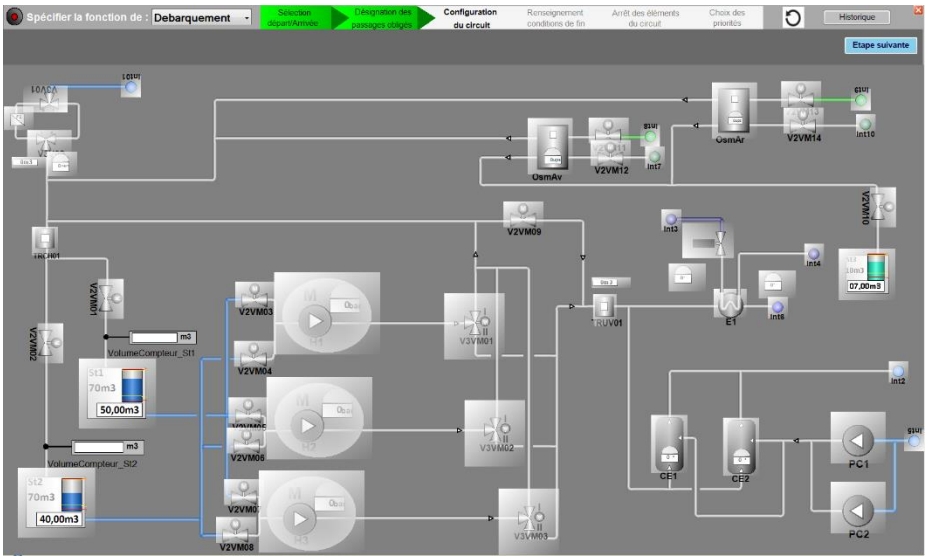


Figure 150 Bouton étape suivante en haut à droite

Pour 7 d'entre eux, ce problème a été rencontré plus d'une fois, soit 53.85%, ce qui témoigne d'un réel problème d'utilisabilité. En effet, même connu, ce problème continue à réapparaître. Par ailleurs, on constate que ce problème réapparaît de moins en moins souvent au fil des fonctions ce qui démontre un effet d'apprentissage (Figure 151).

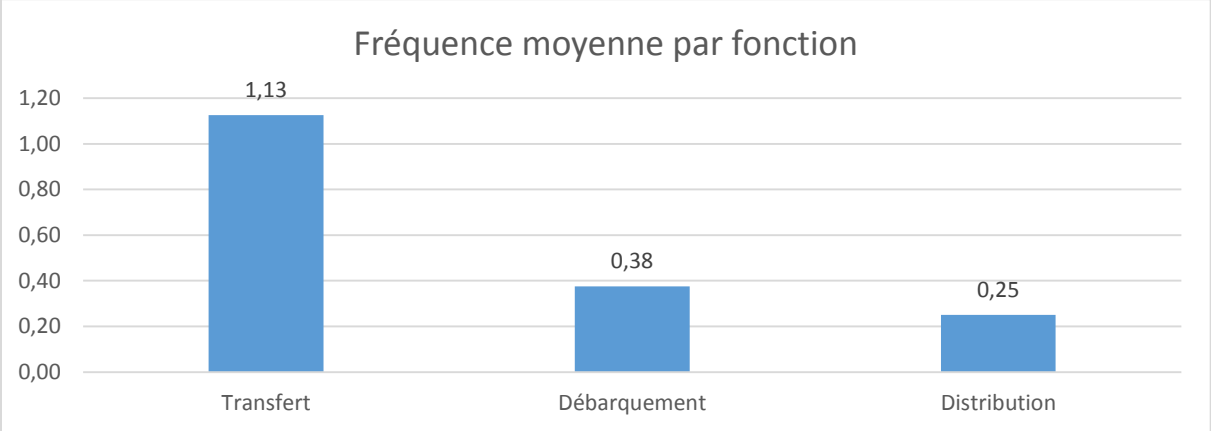


Figure 151 Fréquence moyenne d'apparition du problème d'utilisation du bouton étape suivante par fonction

Le fait de devoir cliquer sur le bouton « étape suivante » après avoir enregistré pour pouvoir accéder à l'étape suivante ne va pas de soi selon les participants :

« *C'est vrai que ça ce n'est pas intuitif.* » (Participant 1).

« *Ça c'est bizarre aussi qu'à chaque fois qu'on clique sur enregistrer il faille cliquer sur étape suivante. On s'attend à ce que ça passe directement à l'étape suivante.* » (Participant 2)

« *Je pensais qu'une fois que j'avais cliqué sur enregistrer je n'avais pas besoin de cliquer sur étape suivante, des choses comme ça. Ensuite la 2eme fois, là on sait où on va donc c'est tout de suite bien plus simple.* » (Participant 5).

En effet, l'animation de la barre de progression est le premier élément trompeur. Elle laisse penser que l'utilisateur est passé à l'étape suivante (Figure 152). Les participants étaient alors souvent tentés de commencer l'étape en essayant d'agir sur le PID.

Figure 152 Apparence de la barre de progression après avoir enregistré

Le deuxième problème de ce bouton est son emplacement, excentré, en haut à droite. En effet, ce n'est pas le premier endroit où les participants posent leur regard. L'analyse les tests laisse penser qu'il arrive que le bouton « étape suivante » ait été remarqué mais que les participants ne comprennent pas qu'ils doivent cliquer dessus.

Le bouton « étape suivante » devrait être supprimé pour que l'utilisation soit plus fluide et plus intuitive pour les participants.

Quitter l'enregistreur

10 participants sur 16 ont eu des difficultés à quitter l'enregistreur pour se rendre dans le rejoueur, soit 62,5% des participants. En effet, pour quitter l'enregistreur, il faut cliquer sur la petite croix rouge en haut à droite de l'écran (Figure 153).

Ce qui pose problème ici, c'est que les participants pensent que ce bouton sert à quitter le logiciel :

« *Parce que c'est vrai que si on dit qu'on ferme bah on va partir et on ne va plus rien faire.* » (Participant 1).

« *En fait je peux quitter comme ça ou ça va… ? J'avais peur que ça ferme totalement le logiciel, (Clic sur la croix) je m'attendais à ce que ça ferme totalement le logiciel.* » (Participant 2).

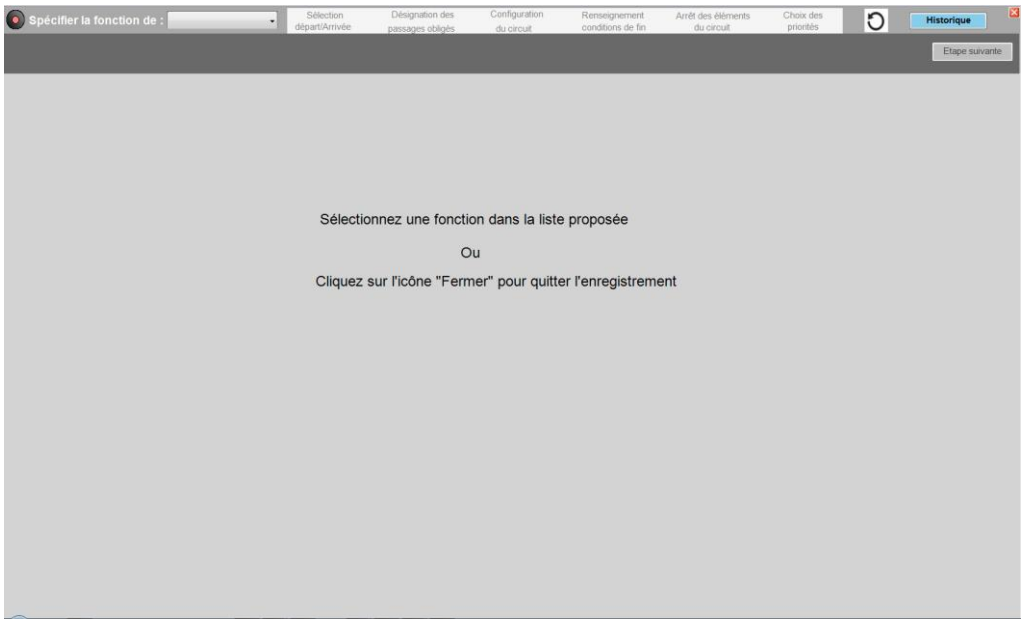


Figure 153 Apparence de l'interface lorsque l'enregistrement d'une fonction est terminé.

Pour éviter ces problèmes, le bouton pourrait être remplacé par un bouton intitulé « quitter l'enregistreur » par exemple. Une deuxième solution serait d'amener directement l'utilisateur à la page d'accueil lorsqu'il termine l'enregistrement d'une fonction mais il ne pourrait pas directement enchaîner l'enregistrement d'une autre fonction comme ici.

Renseigner les conditions d'arrêt

L'étape « Renseignement des conditions de fin » a posé problème à 10 participants sur 16, soit 62,5% des participants (Figure 154).

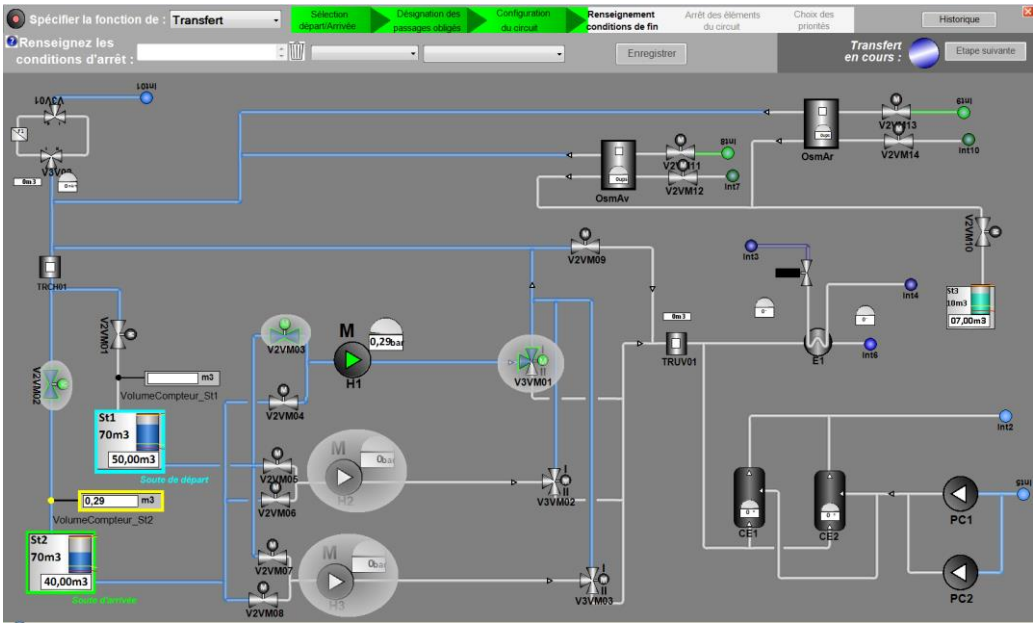


Figure 154 Apparence de l'interface pour l'étape « renseignement des conditions de fin » (Fonction transfert)

Le problème majeur de cette étape est de savoir comment il faut interagir avec la zone de texte. Comme il s'agit d'une zone de texte, les utilisateurs sont tentés d'y écrire :

« *Participant 6 : (Il revient sur la zone de texte et essaye de cliquer dedans.). Ici je veux mettre l'information du compteur. Du coup, euh… (Il positionne sa souris sur le compteur st2, surligné en jaune.). Parce que là je devrais mettre égal à 20m3 (il va dans la première liste, il choisit égal. Puis il clique dans la deuxième liste). Ah non. Ah ! (il revient dans la zone de texte et clique dedans. Ça ne marche pas et il a l'air de ne pas savoir quoi faire). Parce que je voudrais mettre compteur égal à 20m3.*

Expérimentateur : Oui. Alors je vais t'aider. Il faut que tu cliques sur le schéma sur… voilà (il clique sur compteur st2). » (Participant 6).

« *Du coup je clique sur… (il désigne le volume compteur st2 qui est surligné en jaune) ou je dois taper compteur… (il désigne la zone de texte en haut).* » (Participant 10).

L'apparence de ce champ sous forme de zone de texte est trompeuse. Malgré le compteur qui est entouré en jaune et qui clignote, les participants tentent d'écrire le volume qu'ils ont renseigné directement dans la zone de texte. Le comportement de l'interface doit être revu afin de clarifier le mode d'interaction ici.

Le second problème à cette étape est de savoir ce que contiennent les deux listes. En effet, pour quelqu'un qui découvre l'interface, il faut cliquer sur les listes pour savoir ce qu'elles comportent. De plus, si l'on commence à remplir dans le sens de la lecture, de gauche à droite, on ne peut savoir quoi choisir dans la 1ère liste que si l'on sait ce qu'il y a dans la deuxième. Cela implique donc de faire des allers-retours entre les deux listes. Il est possible de parer à ce problème en laissant visible le premier nom de la liste ou bien en ajoutant un titre.

L'animation « [Fonction] en cours »

9 participants sur 16 ont été perturbés par l'animation « [Fonction] en cours », soit 56,25% des participants (Figure 155).

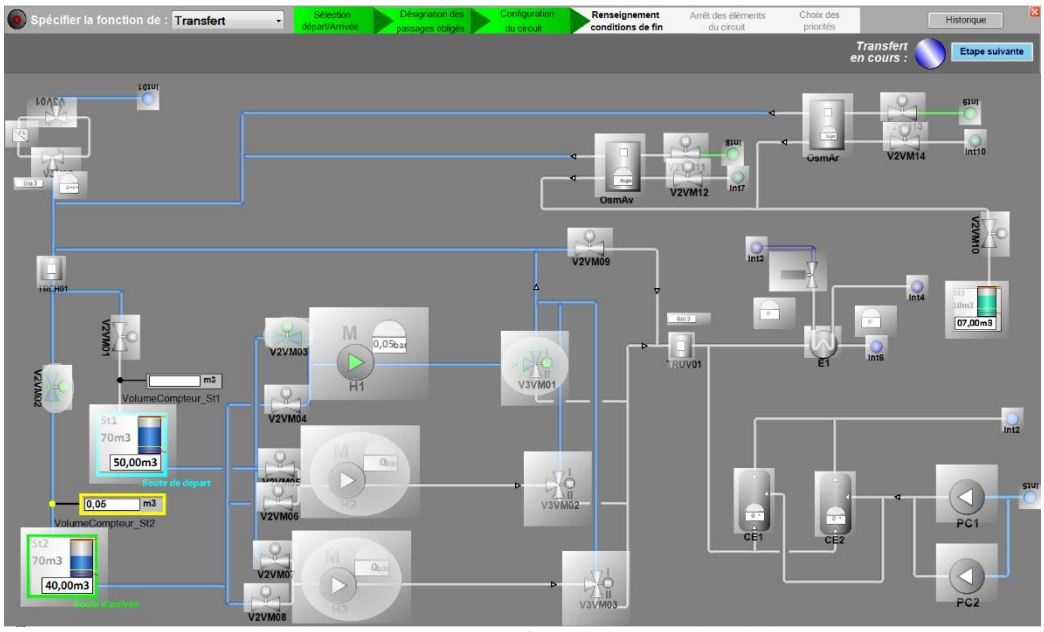


Figure 155 Animation « transfert en cours » en haut à droite

En effet, nous avons pu constater que les participants pensaient qu'il fallait attendre que le transfert (par exemple) ait finit de se réaliser pour pouvoir passer à l'étape suivante :

« *(Ecran avec transfert en cours). Donc là je suppose qu'on va patienter.* » (Participant 2).

« *On attend qu'il aille jusqu'à 20 ? (elle désigne le compteur st2 qui est entouré en jaune et dont les valeurs changent).* » (Participant 4).

« *Transfert en cours. (Attente). Donc là je dois attendre qu'il atteigne 5m3 ?* » (Participant 7).

Cette animation, qui offre un aperçu de ce qui se passera sur l'interface finale, n'apporte aucun élément utile pour l'utilisateur. Au contraire, elle le distrait de sa tâche. Pour cette raison, cette animation devrait être supprimée.

La fenêtre de commande des vannes manuelles (fonction débarquement)

Cette fenêtre de commande a posé plusieurs difficultés. La première, et la plus conséquente, concerne l'identification des voies des vannes manuelles. En effet, 8 participants sur 16 ont eu des difficultés à identifier les différentes voies des vannes manuelles, soit 50% (Figure 156).

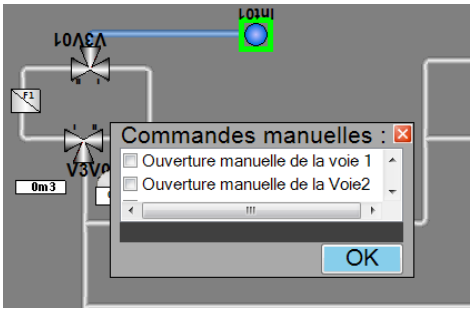


Figure 156 Les vannes manuelles

L'analyse des tests montre que les participants n'arrivent pas à voir les chiffres romains près des vannes sur le schéma PID :

« *Alors ouverture manuelle voie 1. Ouverture manuelle voie 2. Alors c'est quoi la voie 1 et la voie 2 ici ?* » (Participant 3).

« *Participant 4 : C'est laquelle la voie 1 et la voie 2 ?*

Expérimentateur : Il y a des petits chiffres romains normalement sur les sorties.

Participant 4 : Ah oui, donc le carré ça doit être le 2 (elle sélectionne la voie 2). » (Participant 4).

« *Ah oui, je vois les 1 et 2, ils sont un peu petits. Ok.* » (Participant 11).

Pour faciliter l'identification des voies des vannes manuelles, il faut agrandir la police d'écriture pour rendre les chiffres lisibles.

Un deuxième problème concerne le fait que le système laisse la ligne cochée à l'étape d'ouverture lorsque l'utilisateur doit fermer la vanne à l'étape de fermeture des éléments. En effet, avant de cocher « fermeture de la vanne », les participants devaient au préalable décocher « ouverture manuelle de la voie 1/2 ». Durant les tests 4 participants sur 16, soit 25%, l'ont oublié.

Enfin, deux des participants ont fait remarquer que la fenêtre devrait être un peu plus grande pour leur permettre de voir tous les choix d'un coup :

« *C'est dommage ici aussi qu'on doive cliquer là alors que la fenêtre pourrait être un peu plus grande. Ce serait plus intuitif.* » (Participant 2).

« *Il faut que la fenêtre soit un peu plus grande pour qu'on ne soit pas obligé de dérouler la liste pour cocher.* » (Participant 6).

5.2.3.2.4 Questionnaire S.U.S.

Les résultats au questionnaire S.U.S. montre un score d'utilisabilité de 81.56 sur 100 en moyenne (ET = 11,32 ; Min = 55 ; Max = 95), ce qui est encourageant.

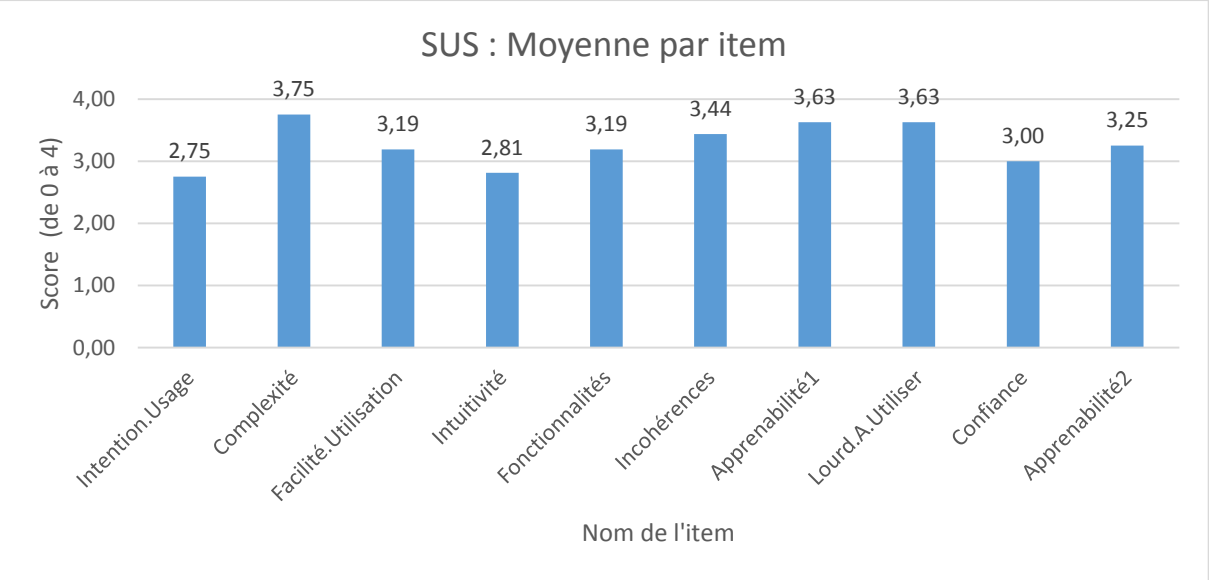


Figure 157 Scores moyens par item du questionnaire S.U.S.

L'analyse item par item (Figure 157) montre que les items les moins bien notés sont les intentions d'usage (2,75 en moyenne), l'intuitivité (2.81 en moyenne) et la confiance (3,00 en moyenne).

Le faible score aux intentions d'usage peut s'expliquer par la difficulté des participants à se projeter dans leur métier futur. En effet, plusieurs participants ont rapporté que cela allait dépendre de leur métier. En ce qui concerne l'intuitivité, cela confirme les problèmes d'utilisabilité relevés dans les sections ci-dessus.

5.2.3.2.5 Questionnaire libre

Nous avons réalisé un nuage de mots à partir des commentaires reçus (Annexe 7). Nous pouvons voir à travers ce nuage de mots que l'interface est le plus souvent qualifiée d'intuitive, simple, claire et facile à utiliser (Figure 158).

Par ailleurs, nous avons recensé et classé les points d'amélioration notés par les participants (Tableau 8) en 3 catégories : correction et prévention des erreurs, guidage et lisibilité/clarté. Dans la première, les participants font allusion à l'impossibilité de corriger ses erreurs. En effet, même si des boutons sont à disposition des participants pour annuler leurs actions, le système n'en tient pas vraiment compte et cela génère des bugs (souvent au niveau du rejeu). Dans la deuxième catégorie, les participants font allusion au bouton « étape suivante » qui a créé beaucoup de difficultés au cours des tests. Enfin, la dernière regroupe différentes suggestions d'amélioration en ce qui concerne la lisibilité de l'interface. On retrouve notamment la fenêtre de commandes des vannes manuelles à agrandir.

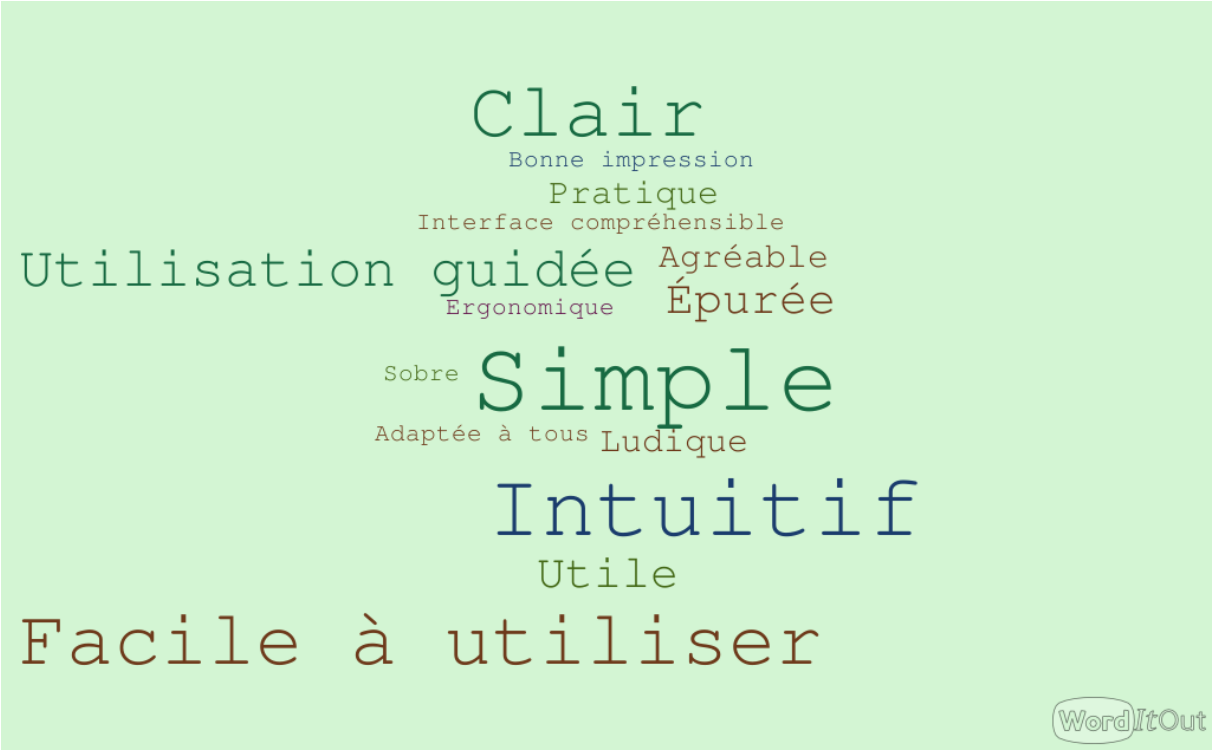


Figure 158 Nuage de mots réalisé à partir des commentaires des participants

Groupement	Commentaires
Correction & Prévention des erreurs	*« Possibilité de revenir en arrière sans faire beuguer l'interface » (Participant 4)* *« Réinitialisation. Annulation des actions » (Participant 6)* *« Correction des différents bugs » (Participant 7)* *« Retour en arrière si erreur » (Participant 8)* *« Ouverture de toutes les vannes avant de pouvoir allumer la pompe » (Participant 13) « Mettre un bouton stop en cas de « bug ». » (Participant 15)* *« Bugs en cas d'erreur de clic où tout doit être recommencé. » (Participant 16)*
Guidage	*« Entourer « étape suivante ». » (Participant 9)* *« Bouton « étape suivante » un peu exilé à droite. Mettre les boutons suivant plus en évidence et le bandeau des instructions/étapes en couleur autre que le gris qui se confond avec le reste » (Participant 10)* *« De prime abord, j'ai eu de légères difficultés à repérer les endroits d'interaction » (Participant 11)* *« Certains points ne sont pas encore très intuitif (la nécessité de re-cliquer sur étape suivante après validation, ou le cadre jaune autour de certains éléments me faisaient penser que j'oubliais quelque chose ou avait fait une erreur) » (Participant 12)* *« Peut-être incorporer un tutoriel en ligne lors de la première utilisation du logiciel » (Participant 14)*
Lisibilité, clarté	*« Encore un peu d'ergonomie de cohérence (menu déroulant…) » (Participant 2)* *« Liste déroulante et « sélectionner tout ». » (Participant 8)* *« Pouvoir ajouter la fonction loupe » (Participant 9)* *« Menu pour vannes manuelles à agrandir pour voir la commande « fermeture », de même pour les différentes options/combinaisons de circuit » (Participant 10)*

Tableau 8 Groupement des commentaires des participants en catégories en ce qui concerne les points à améliorer sur l'interface

5.2.4 Discussion

Les résultats de ces tests utilisateurs sont encourageants même s'il reste encore quelques efforts à fournir concernant l'utilisabilité de l'interface. En effet, les tests ont mis en évidence une quantité importante de bugs, puisque tous les participants ont été concernés au moins une fois par un bug au cours des tests. Souvent, cela entraînait un redémarrage de l'interface, voire même un abandon lorsqu'il s'agissait du rejeu. L'abandon était décidé par l'expérimentateur, tout comme le choix du redémarrage pour finir la fonction. Le système ne permet pas encore une véritable correction des erreurs, ce qui entraîne ces bugs la plupart du temps. Quelques problèmes de guidage ont également été relevés, comme le bouton étape suivante, souvent oublié ou non détecté, ou bien le renseignement des conditions d'arrêt, dont le mode d'interaction n'est pas toujours clair. Les tests utilisateurs montrent que les modifications apportées à l'interface après la première évaluation ont permis d'améliorer l'utilisabilité de l'interface. En effet, malgré les quelques problèmes d'utilisation relevés, nous avons observé une bonne prise en main de l'outil. Les participants l'ont mentionné dans leurs commentaires et le temps passé pour réaliser la spécification des fonctions a diminué dès la deuxième fonction. Par ailleurs, l'interface a été très bien accueillie par les participants. Ils qualifient l'interface d'intuitive, simple, claire et facile à utiliser, ce qui est très positif. Les scores d'utilisabilité recueillis par le biais du questionnaire S.U.S. sont également très encourageants avec une moyenne de 81%. Cela montre que les modifications apportées à l'interface n'ont pas été vaines. En effet, nous avions notamment pour objectif de rendre l'interface plus intuitive

en remplaçant les fenêtres pop-up de guidage par des indices implicites sur l'interface. Ces tests utilisateurs nous ont permis de confirmer l'utilité de notre démarche et de notre outil et également de constater les progrès en ce qui concerne l'utilisabilité.

6 Bilan global des évaluations

Les évaluations réalisées ont donné des résultats encourageants et complémentaires.La *première évaluation* consiste à, dans un premier temps, confronter l'IHM d'EGRC à des utilisateurs expérimentés qui maitrise le système, dans notre démarche de conception. Ce premier test nous a permis de valider l'utilité de notre approche d'utilisation de l'EUD pour la description des spécifications fonctionnelles d'un système. L'utilisabilité (ISO 9241-11 et ISO-13407) de l'outil de spécification a été également prouvée lors de ces entretiens semi-directifs avec les utilisateurs. Néanmoins, nous avons lors de ces tests, détecté quelques problèmes d'utilisabilité sur l'interface. Un audit ergonomique basé sur des méthodes d'expertises, effectué par un ergonome, nous a permis d'apporter des solutions aux problèmes détectés et de dégager d'autres points d'amélioration qui n'ont pas été détectés lors des tests. Les informations issues de ces analyses ont été utilisées pour rendre l'IHM d'EGRC plus intuitive à l'utilisateur.

Suite à cela, l'IHM d'EGRC a été soumise, pour une *deuxième évaluation*, à des tests utilisateurs plus stricts basés sur un questionnaire SUS (*System Usability Scale)*. Le résultat de cette évaluation confirme l'utilité de notre démarche et l'utilisabilité de cette IHM. Cependant, les problèmes relevés lors de ces tests sont pour la plupart intervenue lors du rejeu des fonctions, car l'IHM d'EGRC n'offre pas à l'utilisateur expert de corriger les programmes issus de la généralisation. Nous ouvrons donc la voie à l'introduction d'outils permettant la correction de programmes par utilisateur non-informaticien, dans notre IHM d'EGRC.

Conclusion et perspectives

La conception de systèmes complexes est une activité qui nécessite un grand nombre d'acteurs issus d'horizons techniques très variés. Le nombre des intervenants et la diversité des domaines d'études entraînent des problèmes de cohérence d'ensemble et d'interprétation des spécifications.

Si l'ingénierie système, et en particulier les techniques issues de l'ingénierie dirigée par les modèles, a pu apporter un début de réponse à cette problématique, l'obtention des modèles d'entrée nécessaires au processus de conception reste la principale difficulté, surtout concernant les domaines d'études où les modèles d'entrée sont peu formalisés, et où il faut prendre en compte l'utilisateur.

Les aspects fonctionnels d'un système sociotechnique, parce qu'ils nécessitent la prise en compte du système dans son ensemble et suivant différents points de vue, souffrent de cette lacune.

Appliqués au domaine de la supervision de procédés industriels, nos travaux proposent une réponse à la problématique de l'obtention des spécifications fonctionnelles pour la génération d'une application complète de contrôle-commande. Nous nous inscrivons dans la continuité de travaux précédant (Bignon 2012) qui ont proposé un démonstrateur permettant la génération d'une chaîne de contrôle-commande de bas-niveau, basée sur l'architecture du système à piloter. Nous avons complété ce démonstrateur par une approche plus fonctionnelle du système permettant la création de commandes de niveau coordination ainsi que des dispositifs d'interactions associés.

Rappel des contributions

Les travaux présentés dans cette thèse ont abouti à la proposition d'une démarche permettant de faciliter la conception des modèles de tâches complexes, l'obtention des spécifications fonctionnelles et la conception des systèmes de contrôle-commande. L'objectif principal est de limiter les erreurs liées à l'interprétation des spécifications et de simplifier la conception des systèmes de contrôle-commande.

Tout d'abord, la prise en compte des facteurs humains dans la conception nous a conduits à nous intéresser à la modélisation de l'activité de l'opérateur en supervision, sur le système de contrôle-commande. Les tâches d'un opérateur en supervision sont tellement complexes que leur modélisation devient vite fastidieuse. Nous avons retenu de l'état de l'art que les approches existantes, basées sur les patterns de tâches, permettant de faciliter la conception de tels modèles, manquent de flexibilité. En effet, il n'est pas possible à un expert métier (non-concepteur d'IHM) d'adapter ces patterns. Or, ce sont ces experts qui disposent de toute la connaissance nécessaire pour décrire les tâches des opérateurs sur le système. Il nous a paru important de proposer une démarche permettant à l'expert métier d'adapter les modèles de tâches tout en les simulant, en les vérifiant et en les validant. Notre approche étend la méthode utilisée dans le simulateur de tâches *Prototask* (Lachaume et al. 2012) qui permet de faciliter la vérification et la validation des modèles de tâches par les utilisateurs.

Une fois les modèles de tâches obtenus, nous les avons utilisés pour générer les commandes de haut-niveau du système à concevoir. Pour ce faire, nous avons étudié les approches basées sur la génération des applications interactives à partir des modèles de tâches. Cela nous a

permis d'identifier les verrous qui découlent de ces approches et de remarquer un manque crucial d'informations qui peuvent être issues des spécifications fonctionnelles du système.

Nous nous sommes alors intéressés aux techniques d'EUD (End User Development) permettant aux non-informaticiens de créer des programmes. En s'appuyant sur ces techniques, nous proposons un outil de spécification qui dispose d'une interface d'EGRC (Enregistrement Généralisation Rejeu Correction). Cette interface générée automatiquement, permet à un utilisateur expert (expert métier non-informaticien) en charge de la spécification du système, de décrire plus facilement les spécifications des systèmes complexes. Les spécifications fonctionnelles sont composées de *programmes génériques* et de *configurations possibles* des fonctions du système à concevoir.

Ces apports théoriques nous ont permis de proposer une démarche de conception d'application de contrôle-commande en dix opérations, partant des modèles de tâches, et mettant en œuvre les techniques de l'EUD afin d'obtenir des spécifications fonctionnelles, pour aboutir à la génération d'applicatifs de contrôle-commande.

Notre démarche est fondée sur un pattern de tâches issu de notre retour d'expérience industriel et de l'analyse des activités des opérateurs en supervision, tirée de l'état de l'art.

Une première opération consiste à proposer à l'expert métier, un outil Prototask Editor, lui permettant d'utiliser ses connaissances pour adapter le pattern de tâches de supervision proposé, pour obtenir facilement les différents **modèles de tâches** correspondant aux fonctions du système à concevoir.

Une seconde opération introduit les informations issues des modèles de tâches précédemment obtenus dans un composant EGRC pour obtenir un **modèle EGRC** qui adapté aux spécificités du système à concevoir, et qui permet la mise en œuvre des techniques d'EUD.

Une troisième opération met en œuvre le concept de patron car les informations relatives à la spécification du système et référencées dans la nomenclature issue des travaux de (Bignon 2012), ainsi que le modèle EGRC sont insérés dans un modèle standard qui est vu comme un patron de conception. Cette troisième opération génère ainsi une **IHM d'EG** (Enregistrement Généralisation).

Cette IHM est utilisée dans la quatrième opération, pour capturer les connaissances de l'utilisateur expert (expert métier) sur le système à concevoir. Il est demandé à l'expert métier d'enregistrer deux exemples de spécification pour chacune des fonctions du système à concevoir. Ces exemples sont ensuite généralisés par un module de généralisation liée à l'IHM, pour proposer les **spécifications fonctionnelles généralisées** du système.

Combinées avec les modèles de tâches, les spécifications fonctionnelles sont utilisées dans une cinquième opération, pour générer automatiquement les **interfaces de contrôle** des commandes de haut-niveau.

Ces interfaces de contrôle sont intégrées à l'IHM d'EG pour former une **IHM d'EGRC**, dans une sixième opération.

Une septième opération consiste à utiliser l'IHM d'EGRC pour permettre à l'expert métier lors d'une phase de rejeu de tester, vérifier et valider les spécifications fonctionnelles généralisées précédemment obtenues. À l'issue de cette opération, on obtient les **configurations (**contenues

dans les des spécifications fonctionnelles) **vérifiées et validées** par l'expert métier et la **présentation** des **interfaces de contrôle vérifiée et validée** par l'ergonome.

Une huitième opération consiste à générer les **codes de commande de haut-niveau** (commandes de coordination) à partir des spécifications fonctionnelles.

Une neuvième opération consiste à enrichir la structure et l'architecture de l'IHM basique issue des travaux de (Bignon 2012), par les interfaces de contrôle afin d'obtenir une **IHM complète**.

Une dixième opération consiste à enrichir le programme de commande élémentaire issu des travaux de (Bignon 2012), par les codes de commande de haut-niveau afin d'obtenir un **programme de commande complet**.

L'utilisation des techniques de l'IDM pour générer les différents modèles finaux opérationnels, à partir des informations issues du P&ID et des bibliothèques de haut-niveau (bibliothèque de widgets et bibliothèque de Grafcets), permet alors aux experts de se **concentrer sur leurs cœurs de métier**. L'implémentation de l'outil Prototask Editor et la génération de l'IHM d'EGRC offrent aux experts métiers des moyens leur permettant d'exprimer plus facilement leurs expertises sur le système qu'ils conçoivent. Nous offrons également la possibilité à l'ergonome d'améliorer les interfaces de contrôle générées à travers l'IHM d'EGRC.

L'ensemble de notre démarche a, par ailleurs, été développé, puis intégré à l'outil *Anaxagore* (Bignon 2012), pour le compléter. Pour valider notre démarche, nous l'avons appliquée à un cas d'étude. Nous avons ensuite réalisé deux évaluations afin de vérifier l'utilité de la démarche proposée ainsi que l'utilisabilité de l'IHM d'ERGC générée.

Lors de la première évaluation, l'outil de spécification a été confronté à des utilisateurs expérimentés. L'analyse des résultats nous a permis de valider l'utilité de notre démarche de spécification. En effet, les participants ont obtenus en quelques clics, des spécifications fonctionnelles du système qui leur a été présenté. À travers les commandes de haut-niveau générées à partir de ces spécifications, ils ont pu tester, vérifier et valider les spécifications fonctionnelles obtenues.

Malgré des résultats satisfaisants, cette première évaluation nous a révélé quelques défauts d'utilisabilité de l'interface. Suite à cela, l'outil de spécification a été amélioré, puis soumis à une deuxième évaluation.

La deuxième évaluation a confirmé l'amélioration de l'utilisabilité de l'IHM d'EGRC. Néanmoins, les problèmes relevés lors de l'étape de test, vérification et de validation des spécifications fonctionnelles sont généralement issus des erreurs faites par les participants lors de la spécification. Il est donc essentiel d'élargir notre démarche.

Perspectives

Des perspectives à court terme sont envisageables tant sur la conception des modèles de tâches complexes, que pour l'obtention des spécifications fonctionnelles.

Tout d'abord, l'outil Prototask Editor proposé aux experts métiers pour obtenir facilement les modèles de tâches devra être finalisé, pour être ensuite confronté à ces experts, lors de tests utilisateurs. L'objectif sera de vérifier l'acceptabilité de notre démarche d'adaptation de pattern de tâches par utilisateur expert.

Concernant la spécification fonctionnelle des systèmes complexe, l'IHM d'EGRC présentée dans ce mémoire permet à l'expert métier de corriger les configurations et à l'ergonome de corriger les présentations. Seules les corrections des configurations et des présentations ne suffisent pas pour obtenir des spécifications fonctionnelles vérifiées et validées. Même si l'expert métier (utilisateur expert) possède une bonne connaissance du système qu'il conçoit, il peut faire des erreurs en décrivant son fonctionnement. Ces erreurs contenues dans les programmes enregistrés qui sont en réalité, les actions de l'utilisateur sur le système, sont propagées aux programmes généralisés. Il est donc essentiel de lui permettre de corriger ces erreurs dans le programme. La **correction des programmes généralisés** (représentant les différents dialogues) **par un expert non-informaticien** reste un problème très complexe que nous n'avons traité dans cette thèse. Il s'agira de compléter notre outil de spécification par une interface basée sur les techniques d'EUD permettant à l'expert métier de corriger les programmes généralisés et/ou enregistrés.

Nos travaux ouvrent enfin d'autres perspectives intéressantes qui concernent l'extension de notre flot de conception par la connexion d'un simulateur, la prise en compte des modes et priorités de fonctionnement dans notre démarche de génération d'applicatifs de contrôle-commande, et l'évolution de notre démonstrateur d'outils. Toutes ces perspectives sont discutées dans les paragraphes suivants.

- *Connexion de l'IHM d'EGRC à un simulateur*

L'IHM d'EGRC présentée dans ce mémoire n'interagit pas avec le reste de la chaine de contrôle-commande (partie commande et procédé) faute de simulateur. Un simulateur est un prototype virtuel qui rend compte du comportement réel du procédé physique.

Le simulateur issu des travaux de (Prat et al. 2015) devra être intégré dans notre démarche de spécification.

L'ajout d'un simulateur permettra d'obtenir des commandes de haut-niveau qui reflètent le fonctionnement réel du système, tant en situation normale, qu'anormale. Suite à cela, notre flot de conception proposée à la Figure 1 Chapitre 4 pourra être modifié pour permettre de générer les codes de commande de haut-niveau, en même temps que les interfaces de contrôle. De cette façon, nous pourrons assurer la cohérence entre ces deux modèles qui doivent s'échanger des données. Cependant, une opération supplémentaire devra être intégrée à notre flot pour mettre à jour les commandes de haut-niveau, après l'opération de test, vérification et correction.

L'objectif de la connexion d'un simulateur à notre outil de spécification serait d'une part, de s'assurer que les configurations spécifiées permettent de réaliser la fonction à laquelle elles sont associées. Et d'autre part, de s'assurer que ces configurations non seulement permettent d'atteindre une configuration réalisant la fonction, mais aussi qu'elles respectent les contraintes de fonctionnement et de sécurité du système. De plus, la cohérence sémantique entre le programme de commande complet et l'IHM de supervision complète générés, pour la demande de réalisation d'une fonction, pourrait être vérifiée.

- *Prise en compte des modes d'exploitation et des priorités des fonctions*

L'IHM de supervision complète et le programme de commande complet générés ne prennent pas en compte les modes d'exploitation des fonctions et la gestion de priorités entre les fonctions en cas d'exécution simultanées.

Concernant les modes d'exploitation, il reste à étudier comment spécifier précisément les modes de fonctionnement des fonctions du système à concevoir, pour les intégrer dans la démarche de génération d'applications de contrôle-commande. La spécification des modes de fonctionnement ne dépend pas que de l'expert métier. Elle peut nécessiter l'intervention des opérateurs, des clients, etc… et reste non formalisée. À moindre échelle, on retrouve des erreurs d'interprétation des spécifications de modes qui peuvent être propagées dans la conception, du fait de la différence de culture technique des intervenants. Il deviendra alors important de s'intéresser à la gestion de ces modes de fonctionnement dans la génération des applications contrôle-commande.

Concernant les priorités des fonctions, la démarche de spécification fonctionnelle proposée dans cette thèse se concentre sur la spécification fonction par fonction, en permettant à l'expert métier d'attribuer une priorité à chaque fonction spécifiée. Cependant, dans la conception des systèmes de contrôle-commande, cette spécification n'est pas suffisante pour tenir compte des activités réelles de la supervision. Les lois qui régissent le passage d'une fonction à une autre ou la réalisation d'une fonction dépendent de l'état du système et de la priorité entre les fonctions. Au niveau du système, on ne s'intéresse pas à la réalisation d'une seule fonction, mais de plusieurs. Tout le problème dans l'exécution simultanée de plusieurs fonctions, consiste à trouver le bon ordonnancement vis-à-vis des contraintes pesant sur le système. Ces contraintes sont issues de la dynamique du procédé physique contrôlé. Il nous parait alors intéressant d'étudier les approches d'ordonnancement pour la gestion des priorités, permettant de gérer les ressources de façon efficace. En effet, la nécessité d'ordonnancer apparaît, généralement, dès que plusieurs tâches peuvent être actives simultanément.

Par ailleurs, l'introduction de l'ordonnancement ne fera pas abstraction des questions de sûreté de fonctionnement. Le traitement des problèmes de sûreté de fonctionnement est en cours avec l'implémentation d'un module de reconfiguration dynamique.

La reconfiguration dynamique consiste à trouver une configuration alternative en cas de défaillance d'un élément du système. L'introduction des approches d'ordonnancement pour la gestion des priorités dans le processus de la reconfiguration dynamique devrait permettre de gérer le parallélisme entre les fonctions, en cas d'aléas ou en cas de commandes de haut-niveau conflictuelles.

- *Évolution de notre démonstrateur d'outils*

Globalement, le démonstrateur d'outils de conception développé au cours de nos travaux offre plusieurs perspectives d'évolution quant aux dispositifs d'implantation et aux plateformes d'exécution.

L'intégration dans le processus de génération de la propriété de plasticité (Oliveira, Dupuy-chessa, et Calvary 2015) permettrait d'obtenir des interfaces adaptatives qui peuvent être déployées facilement. Une interface adaptative est une interface qui se transforme, en fonction des évolutions du contexte. Trois types adaptations sont envisageables : l'adaptation au profil de l'utilisateur, l'adaptation à la plateforme d'exécution et l'adaptation à l'environnement applicatif. Les techniques d'adaptation utilisées devront garantir le respect des propriétés du système et des interfaces.

Pour rendre notre démarche complètement générique, il est nécessaire de s'intéresser aux plateformes d'exécution. Notre processus de génération s'appuie, à ce jour, sur un choix de

logiciels de contrôle-commande effectué par (Bignon 2012). Ces logiciels servent d'une part, à décrire les éléments de la bibliothèque standard, puis le modèle standard d'IHM. Et d'autre part, ils servent à interpréter les IHM et les programmes de commande générés. L'utilisation de nouveaux outils conduira à la modification de quelques-unes de nos transformations de modèle. Il serait intéressant d'explorer des pistes pouvant permettre d'une part de partir de modèles de type PIM (Plateform Independant Model) pour ensuite les adapter à différents logiciels afin d'obtenir des PSM (Plateform Specific Model). D'autre part, les approches de rétro-ingénierie (Rugaber et Stirewalt 2004) devraient permettre d'adapter les applications générées à d'autres plateformes.

Annexes

Références bibliographiques

Abed, Mourad. 1990. « Contribution à la modélisation de la tâche par des outils de spécification exploitant les mouvements oculaires: appli-cation à la conception et l'évaluation des Interfaces Homme-Machine. » Valenciennes.

———. 2001. *Méthodes et modèles formels et semi-formels pour la conception et l'évaluation des systèmes homme-machine*.

Abraham, Robin, et Martin Erwig. 2007a. « UCheck: A spreadsheet type checker for end users. » *Journal of Visual Languages and Computing* 18: 71-95. doi:10.1016/j.jvlc.2006.06.001.

———. 2007b. « GoalDebug: A spreadsheet debugger for end users. » In *Proceedings - International Conference on Software Engineering*, 251-60. doi:10.1109/ICSE.2007.39.

ABRIAL, JEAN-RAYMOND. 1996. *The B-Book , Assigning Programs to Meanings*. Cambridge: University Press.

AFNOR X50-151. « "Management par la valeur et ses outils, analyse fonctionnelle, analyse de la valeur, conception a objectif designe". » French National Standards, Ed.

Aït-Ameur, Yamine, Patrick Girard, et Francis Jambon. 1998. « A Uniform approach for the Specification and Design of Interactive Systems: the B method. » *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)* Proceeding: 333-52. http://www.lisi.ensma.fr/ftp/pub/documents/papers/1998/1998-dsvis-yaa_pg_fj.pdf.

Alexander, Christopher. 1979. *The Timeless Way of Building*. *New York Oxford University Press*. New York: Oxford University Press. doi:10.1080/00918360802623131.

Alexander, Christopher, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fikdahl-King, et Shlomo Angel. 1977. *A Pattern Language: Towns, Buildings, Construction*. *Leonardo*. Vol. 14. Oxford University Press. doi:10.2307/1574526.

Allegre, Willy. 2012. « Flot de conception dirigée par les modèles pour la commande et la supervision de systèemes domotiques d ' assistance Willy All ` To cite this version : HAL Id : tel-00803402 Willy Allègre Flot de conception dirigé par les modèles pour la commande e. »

Ansi/Isa. 1992. *Instrumentation Symbols and Identification ANSI/ISA-5.1*. Vol. 1984.

Apple. 2013. « Apple Style Guide », nº April: 1-197. https://help.apple.com/asg/mac/2013/ASG_2013.pdf.

Arzapalo, Denisse Yessica Muñante. 2014. « Une approche basée sur l 'Ingénierie Dirigée par les modèles pour identifier, concevoir et évaluer des aspects sécurité. »

Bainbridge, Lisanne, et Michael C Dorneich. 1999. « Processes Underlying Human Performance. » *Handbook of aviation human factors*, 107-71.

Bara, Catalin, Dumitru Popescu, et Florin Gheorghe Filip. 2013. *SCADA system for ship stabilization designed using an expert system*. *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Vol. 6. IFAC. doi:10.3182/20130911-3-BR-3021.00014.

Bâra, Cătălin, Dumitru Popescu, et Ciprian Lupu. 2012. « Expert System for generating SCADA configurations with practical applications in the naval industry. » In *In System Theory, Control and Computing (ICSTCC), 2012 16th International Conference*, 1-6.

BARTHET, Marie-France. 1988. *logiciels interactifs et ergonomie modèles et méthodes de conception*. Informatiq. (Bordas, Paris). http://interaction2.free.fr/Ouvrages/Barthet1988/barthet1988.pdf.

Bastien, Christian. 1991. *Validation de criteres ergonomiques pour l'evaluation d'interfaces utilisateurs*. https://hal.inria.fr/inria-00075133.

Bastien, Christian J M, et Dominique L Scapin. 1993. « Ergonomic Criteria for the Evaluation of Human-Computer Interfaces @BULLET Critères Ergonomiques pour l'Évaluation d'Interfaces Utilisateurs. »

Bayle, E., R. Bellamy, G. Casaday, T. Erickson, S. Fincher, B. Grinter, et C. Potts. 1998. « Putting It All Together : Pattern Languages for Interaction Design : Putting it all together: towards a pattern language for interaction design: A CHI 97 workshop. » In *CHI 97 workshop*, 30:17-23. ACM SIGCHI Bulletin. doi:10.1145/1120212.1120357.

Beckwith, Laura, Margaret Burnett, Susan Wiedenbeck, Curtis Cook, Shraddha Sorte, et Michelle Hastings. 2005. « Effectiveness of end-user debugging software features. » In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*, 10. Portland, Oregon, USA: ACM. doi:10.1145/1054972.1055094.

Berti, Silvia, Giulio Mori, Fabio Paternò, et Carmen Santoro. 2003. « An Environment for Designing and Developing Multi- Platform Interactive Applications. » In *HCI Italy*, 7-16. Italy. https://www.researchgate.net/profile/Fabio_Paterno/publication/228610398_An_environment_for_designing_and_developing_multi-platform_interactive_applications/links/09e4151063833b3b5d000000.pdf.

Bévan, Romain. 2013. « Approche composant pour la commande multi-versions des systèmes transitiques reconfigurables. » http://www.theses.fr/2013LORIS311.

Bignon, Alain. 2012. « Génération conjointe de commandes et d ' interfaces de supervision pour systèmes sociotechniques reconfigurables. » UBS.

Boehm, Barry W, T R W Defense, Systems Group, Harry W Boehm, T R W Defense, et Systems Group. 1988. « A Spiral Model of Software Development and Enhancement. » *Computer* 21 (1): 61-72. doi:10.1109/2.59.

Boucher, Amélie. 2009. *Ergonomie web : pour des sites web efficaces*. Édité par Eyrolles. 2e éd. Eyrolles. http://www.eyrolles.com/Informatique/Livre/ergonomie-web-9782212121582.

Boulhic, Laurianne, Alain Bignon, et Fabien Silone. 2016. « Évaluation expérimentale du codage couleur pour une interface de supervision. »

Bovell, Charles. R., Richard .J. Carter, et Michael. G. Beck. 1998. *Nuclear power plant control room operator control and monitoring tasks*. Oak Ridge, TN. doi:10.2172/663388.

Breedvelt-schouten, M., Ilse, Fabio Paternò, et C Severijns. 1997. « Reusable structures in task models. » *Design, Specification and Verification of Interactive Systems' 97*, 225-39.

Brooke, John. 1996. « SUS : a "quick and dirty" usability scale ». In *Usability Evaluation In Industry*, édité par Patrick W. Jordan, Thomas B., Ian Lyall McClelland, et Bernard Weerdmeester, 4-7. CRC Press.

Broy, M., C. Facchi, et R. Grosu. 1993. *The requirement and design specification language SPECTRUM--an informal introduction*. Munchen, Germany.

Brusilovsky, Peter. 1998. « Methods and Techniques of Adaptive Hypermedia. » In *Adaptive Hypertext and Hypermedia*, 1-43. Dordrecht: Springer Netherlands. doi:10.1007/978-94-017-0617-9_1.

Burnett, Margaret, et Christopher Scaffidi. 2011. *End-User Development.* 2e éd. Interaction Design foncation. https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/end-user-development.

Caffiau, Sybille. 2010. « Approche dirigée par les modèles pour la conception et la validation des applications interactives : une démarche basée sur la modéelisation des tâches. »

Calvary, Gaëlle, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, et Jean Vanderdonckt. 2003. « A Unifying Reference Framework for multi-target user interfaces. » *Interacting with Computers* 15: 289-308. doi:10.1016/S0953-5438(03)00010-9.

Cao, Jill, Kyle Rector, Thomas H. Park, Scott D. Fleming, Margaret Burnett, et Susan Wiedenbeck. 2010. « A debugging perspective on end-user mashup programming. » *Proceedings - 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2010*, 149-56. doi:10.1109/VLHCC.2010.29.

Caumont, Daniel. 2010. *Les études de marché*. Édité par Dunod. 4e édition. http://numerique.dunod.com/85536/Les-etudes-de-marche---4e-edition.ebook.

Chintakovid, Thippaya, Susan Wiedenbeck, Margaret Burnett, et Valentina Grigoreanu. 2006. « Pair collaboration in end-user debugging. » *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006*, 3-10. doi:10.1109/VLHCC.2006.36.

Ciapessoni, Emanuele, Alberto Coen-Porisini, Ernani Crivelli, Dino Mandrioli, Angelo Morzenti, et Piergiorgio Mirandola. 1999. « From formal models to formally-based methods: an industrial experience. » *ACM Transactions on Software Engineering and Methodology (TOSEM)* 8: 79-113. doi:10.1145/295558.295566.

Clarke, Edmund M., et Jeannette M. Wing. 1996. « Formal Methods: State of the Art and Future Directions. » *ACM Computing Surveys* 28 (4): 626-43.

Conte, Agnès, Mounia Fredj, Jean-Pierre Giraudin, et Dominique Rieu. 2001. « P-Sigma : un formalisme pour une représentation unifiée de patrons. » *XIXème Congrès INFORSID*, 67-86. http://liris.cnrs.fr/inforsid/sites/default/files/a366c1YfHw5cvgN2I.pdf.

Cook, D., et S.K Das. 2004. *Smart environments: Technology, protocols and applications*. John Wiley & Sons.

Coutaz, Joelle, Gaelle Calvary, Alexandre Demeure, et Lionel Balme. 2012. « Systèmes interactifs et adaptation centrée utilisateur: la plasticité des Interfaces Homme-Machine. » In *Informatique et Intelligence ambiante : des capteurs aux applications*, Hermes Sci, 1-57. http://hal.archives-ouvertes.fr/hal-00765504/.

Cypher, Allen. 1991. « Eager: Programming Repetitive Tasks by Example. » In *Human Factors in Computing Systems (CHI'91)*, 33-39. New Orleans, Louisiana: ACM/SIGCHI.

Demuynck, M., et B. Meyer. 1979. « Les Langages de spécification. » *Bulletin de la Direction des Etudes et Recherches d'Electricité de France, Série C*, no 1: 39-59.

Fagan, M. E. 1976. « Design and code inspections to reduce errors in program development. » *IBM Systems Journal* 15: 182-211. doi:10.1147/sj.153.0182.

Fanet, Hervé. 1997. « Contrôle-commande et Instrumentation. » In *Instrumentation et contrôle-commande des installations nucléaires*.

Ferrarini, L., A. Dedè, P. Salaun, T. Dang, et G. Fogliazza. 2009. « Domain specific views in model-driven embedded systems design in industrial automation. » In *7th IEEE International Conference on Industrial Informatics*, 702-7.

Fiorèse, Serge, et Jean-Pierre Meinadier. 2012. *Découvrir et comprendre l'Ingénierie Système*. *AFIS*. CEPADUES -. Toulouse, France: COLLECTION AFIS. www.afis.fr.

Fischer, Gerhard, et Andreas Girgensohn. 1990. « End-user modifiability in design environments. » *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, 183-92. doi:10.1145/97243.97272.

Frizon De Lamotte, Florent. 2006. « Proposition d'une approche haut niveau pour la conception, l'analyse et l'implantation des systèmes reconfigurables. » *Lester* PhD thesis.

Gaffar, A., D. Sinnig, A. Seffah, et P. Forbrig. 2004. « Modeling patterns for task models. » *Proceedings of the 3rd annual conference on Task models and diagrams* 2 (1): 99-104. doi:10.1145/1045446.1045465.

Gamma, Erich, Richard Helm, Ralph Johnson, et John Vlissides. 1995. *Design Patterns – Elements of Reusable Object-Oriented Software*. *A New Perspective on Object-Oriented Design*. KevinZhang. doi:10.1093/carcin/bgs084.

Gervais, Frederic. 2004. « EB4 : Vers une méthode combinée de spécification formelle des système d'information. » Université de SHERBROOKE, Quebec Canada.

Girard, Patrick. 1992. « Environnement de Programmation pour Non-Programmeur et Paramétrage en Conception Assistée par Ordinateur : le système LIKE. » *LISI/ENSMA*. Université de Poitiers.

Girard, Patrick, Guillaume Patry, Guy Pierra, et Jean-Claude Potier. 1997. « Deux exemples d'utilisation de la Programmation par Démonstration en Conception Assistée par Ordinateur. » *Revue Internationale de CFAO et d'informatique graphique* 12 (November): 169-88. http://www.lisi.ensma.fr/ftp/pub/documents/papers/1997/1997-micad-girard.pdf.

Goubali, Olga, Alain Bignon, Pascal Berruet, Patrick Girard, et Laurent Guittet. 2014. « Anaxagore , un exemple d ' ingénierie dirigée par les modèles pour la supervision industrielle . », 58-65.

Granlund, Åsa, Daniel Lafrenière, et David a. Carr. 2001. « A Pattern-Supported Approach to the User Interface Design Process. » In *HCI International 2001 9th International Conference on Human-Computer Interaction*, 1:282-86. http://www.sm.luth.se/csee/csn/publications/HCIInt2001Final.pdf.

Green, Mark. 1986. « A survey of three dialogue models. » *ACM Transactions on Graphics* 5 (3): 244-75. doi:10.1145/24054.24057.

Halbert, Daniel. 1993. « SmallStar : Programming by Demonstration in the Desktop Metaphor. » In *Watch What I Do : Programming by Demonstration*, édité par Allen Cypher, 102-23. Cambridge, Massachusuetts: The MIT Press.

Hartson, H. Rex, et Deborah Hix. 1989. « Human-computer interface development: concepts and systems for its management. » *ACM Computing Surveys* 21: 5-92. doi:10.1145/62029.62031.

Hinckley, Ken, R J K Jacob, Colin Ware, et J O Wobbrock. 2014. « Input/Output Devices and Interaction Techniques. » *Dynamical systems with applications using MATLAB* 53: 1-79. doi:10.1017/CBO9781107415324.004.

Hugues, Anne-marie. 2002. « DIFFERENTS MODELES DE CYCLE DE VIE. » *http://users.polytech.unice.fr/~hugues/GL/chapitre2.pdf*. http://users.polytech.unice.fr/~hugues/GL/chapitre2.pdf.

Hutchinson, John, Jon Whittle, Mark Rouncefield, et Steinar Kristoffersen. 2011. « Empirical assessment of MDE in industry. » In *2011 33rd International Conference on Software Engineering (ICSE)*, 471-80. doi:10.1145/1985793.1985858.

IEC. 2003. *IEC 61131-3: Programming industrial automation systems: Concepts and programming languages, requirements for programming systems, decision-making aids*. *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. doi:10.1007/978-3-642-12015-2.

IEEE Compunter Society. 1998. « Recommended Practice for Software Requirements Specifications. » In *Software Engineering Standards Committee, and IEEE-SA Standards Board.*, 830:31. Institute of Electrical and Electronics Engineers. doi:10.1109/ieeestd.1998.88286.

Jacob, R. J., Q. Limbourg, et Jean Vanderdonckt. 2005. « Computer-aided Design of User Interfaces IV. » In *Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004*. Springer Science & Business Media.

Jalila, A., et D.J. Mala. 2014. « Object-oriented model-based specification languages: a comparison. » *ACM* 39 (5): 1-4.

Jaulent, Patrick. 1994. *Génie logiciel, les méthodes Texte imprimé : SADT, SA, E-A, SA-RT, SYS_P_O, OOD, HOOD...* Édité par Armand Colin. 3ème éd. Paris.

Jézéquel, J., S. Gérard, et B. Baudry. 2006. « Le génie logiciel et l ' IDM : une approche unificatrice par les modèles. » In *L'ingénierie dirigée par les modèles*, 54-69.

Johnson, Peter, Stephanie Wilson, Panos Markopoulos, et James Pycock. 1993. *ADEPT -Advanced Design Environment for Prototyping with Task Models*.

Jones, Cliff. B. 1980. *Software development : a rigorous approach*. Prentice/Hall International.

Kahn, Ken. 2000. « Generalizing by Removing Detail : How Any Program Can Be Created by Working with Examples. » *Morgan Kaufman Publishers*, 1-17. http://www.toontalk.com/Papers/yourwish.pdf.

Khalil, Carine. 2011. « Les méthodes "agiles" de management de projets informatiques : une analyse "par la pratique". » Télécom Paris Tech.

Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, et J. Irwin. 1997. « Aspect-oriented programming. » In *European conference on object-oriented programming*, 220-42. Springer Berlin Heidelberg.

Kluge, A. 2014. « Controlling Complex Technical Systems: The Control Room Operator's Tasks in Process Industries. » *The Acquisition of Knowledge and Skills for Taskwork and Teamwork to Control Complex Technical Systems*, 11-47.

Ko, Andrew J, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, et al. 2011. « The state of the art in end-user software engineering. » *ACM Computing Surveys* 43: 1-44. http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=70341636&site=ehost-live.

Ko, Andrew J, et Brad a Myers. 2003. « Development and Evaluation of a Model of Programming Errors Development and Evaluation of a Model of Programming Errors », nº 0.

Ko, Andrew J., B.a. Myers, et H.H. Aung. 2004. « Six Learning Barriers in End-User Programming Systems. » *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, 199-206. doi:10.1109/VLHCC.2004.47.

Kolski, Christophe. 1993. « Exemple de développement d'un outil d'ingénierie pour la conception d'interface: le système ERGO CONCEPTOR. » In *ingénierie des interfaces homme machine conception et évaluation*, Hermes, 289. Paris.

———. 1997. *Interfaces Homme-Machine, application aux systèmes industriels complexes*. Hermès.

Kolski, Christophe, et Houcine Ezzedine. 2003. « Conception et évaluation des IHM de supervision : éléments méthodologiques. » *Revue Génie Logiciel,* 65: 2-11. file:///C:/ETAT_DE_L'ART/SPECIFICATIONS/GL-Kolski-Ezzedine-2003 PDF (1) (1).pdf.

Kolski, Christophe, Houcine Ezzedine, et Mourad Abed. 2001. « Cycle de développement du logiciel : des cycles classiques aux cycles enrichis sous l'angle des interactions homme-machine. » In *Analyse et conception de l'IHM, Interaction Homme-Machine pour les S.I.*, Hermès, 2-11. Paris.

Krieg-Brückner, B. 1990. « PROgram development by SPECification and TRAnsformation. » *echnique et science informatiques (TSI)* 9: 134-49.

Lachaume, Thomas, Patrick Girard, Laurent Guittet, et Allan Fousse. 2012. « ProtoTask, new task model simulator. » *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7623 LNCS: 323-30. doi:10.1007/978-3-642-34347-6_24.

Lachaume, Thomas, Laurent Guittet, Patrick Girard, et Allan Fousse. 2014. « Task model simulators : a review. » *Journal d'Ineraction Personne-Système* 3.

Lajnef, M., M. B. Ayed, et Christophe Kolski. 2005. « Convergence possible des processus du data mining et de conception-évaluation d'IHM : adaptation du modèle en U. » In *17th Conference on l'Interaction Homme-Machine*, 243-46.

Lallican, JL. 2007. « Proposition d'une approche composant pour la conception de la commande des systèmes transitiques. » Université de Bretagne-Sud Lorient.

Le Vie, DS, et S. Donald. 2000. « Understanding Data Flow Diagrams. » In *Proceedings of the Annual Conference on Society for Technical Communication*, 396-401. http://ratandon.mysite.syr.edu/cis453/notes/DFD_over_Flowcharts.pdf.

Leitão, Paulo. 2009. « Agent-based distributed manufacturing control : A state-of-the-art survey. » *Engineering Applications of Artificial Intelligence* 22 (7): 979-91.

Leshed, Gilly, Eben M. Haber, Tara Matthews, et Tessa Lau. 2008. « CoScripter : Automating & Sharing How-To Knowledge in the Enterprise. » *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1719-28. doi:10.1145/1357054.1357323.

Lewandowski, Arnaud, Grégory Bourguin, et Jean-Claude Tarby. 2007. « De l'Orienté Objet à l'Orienté Tâches – Des modèles embarqués pour l ' intégration et le traçage d'un nouveau type de composants. » *Revue d'Interaction Homme-Machine* 8: 1-33.

Lieberman, Henry. 2001. *Your Wish is my command*. Cambridge, MA, USA: Morgan Kaufmann. http://web.media.mit.edu/~lieber/Your-Wish-Intro.html.

Lieberman, Henry, Fabio Paternò, Markus Klann, et Volker Wulf. 2006. « End-user development: An emerging paradigm. » *End User Development* 9: 1-8. doi:10.1007/1-4020-5386-X_1.

Lin, James, Mark W Newman, Jason I Hong, et James a Landay. 2001. « DENIM: An Informal Tool for Early Stage Web Site Design. » *Proceedings of CHI 2001, ACM Conference on Human Factors in Computing Systems*, 205-6. doi:http://doi.acm.org/10.1145/634067.634190.

Liu, Q., K. Nakata, et K. Furuta. 2002. « Display design of process systems based on functional modelling. » *Cognition, Technology & Work* 4 (1): 48-63.

Long, John, et I. Denley. 1990. « Evaluation for patrice. » In *Ergonomics society annual conference*.

Ltifi, Hela. 2011. « Démarche centrée utilisateur pour la conception de SIAD basés sur un processus d ' Extraction de Connaissances à partir de Données , Application à la lutte contre les infections nosocomiales. » l'Université de Valenciennes et du Hainaut-Cambrésis, France et l'Université de Sfax, Tunisie.

Luyten, Kris. 2004. « Dynamic user interface generation for mobile and embedded systems with model-based user interface development. » *Transnationale Universiteit Limburg*. http://research.edm.uhasselt.be/kris/research/phd/v1.pdf.

Luyten, Kris, Tim Clerckx, Karin Coninx, et Jean Vanderdonckt. 2003. « Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. » In *DSV-IS'2003*, 203-17. Funchal, Madeira Island, Portugal: Springer-Verlag.

Marca, D A, et C L McGowan. 1987. « SADT: structured analysis and design technique. »

Marca, David., et Clement L. McGowan. 1988. *SADT : structured analysis and design technique*. McGraw-Hill.

Martinie, Célia. 2011. « Une approche à base de modèles synergiques pour la prise en compte simultanée de l'utilisabilité, la fiabilité et l'opérabilité des systèmes interactifs critiques. » université de Toulouse. doi:10.1017/CBO9781107415324.004.

McAvinew, Thomas, et Raymond Mulley. 2004. *Control System Documentation. The Instrumentation, Systems, and Automation Society*. Library of Congress.

McDaniel, Richard G., et Brad A. Myers. 1999. « Getting More Out Of Programming by Demonstration. » In *Human Factors in Computing Systems (CHI'99)*, édité par Marian G Williams et Mark W Altom, 442-49. Pittsburg: ACM/SIGCHI.

McDermid, John, et Knut Ripken. 1984. *life cycle support in the ada environment*. New York, NY, USA: Cambridge University Press. https://books.google.fr/books?hl=fr&lr=&id=eDY8AAAAIAAJ&oi=fnd&pg=PA1&dq=life+cycle+support+in+the+ada+environment&ots=9WIE8PpYFW&sig=31yl2xOEcZcQVM01TFysg0ckgj0#v=onepage&q=life cycle support in the ada environment&f=false.

Meixner, Gerrir, Fabio Paternò, et Jean Vanderdonckt. 2011. « Past, Present, and Future of Model-Based User Interface Development. »

Meixner, Gerrir, Fabio Paterno', et Jean Vanderdonckt. « Past, Present, and Future of Model-Based User Interface Development. »

Meixner, Gerrit, Gaëlle Calvary, et Joëlle Coutaz. 2013. « Introduction to Model-Based User Interfaces. » *Universal Access in the Information Society*. doi:10.1007/s10209-012-0283-y.

Mens, Tom, Krzysztof Czarnecki, et Pieter Van Gorp. 2006. « A Taxonomy of Model Transformations. » *Theoretical Computer Science*, 125-42. doi:10.1016/j.entcs.2005.10.021.

Mens, Tom, Carine Lucas, et Patrick Steyaert. 1998. « Supporting Reuse and Evolution of UML Models. » In *International Conference on the Unified Modeling Language*, 378-98. Springer Berlin Heidelberg.

Mesli-kesraoui, Soraya, Djamal Kesraoui, Flavio Oquendo, Alain Bignon, Armand Toguyeni, et Pascal Berruet. 2016. « Formal Verification of Software-intensive Systems Architectures described with Piping and Instrumentation Diagrams. » *European Conference on software Architecture*, 1-16.

Mesli-kesraoui, Soraya, Armand Toguyeni, Alain Bignon, Flavio Oquendo, Djamal Kesraoui, et Pascal Berruet. 2016. « Formal and Joint Verification of Control Programs and Supervision Interfaces for Socio-technical Systems Components. » *Ifac Hms*, 1-6.

Microsoft. 2014. *STYLE GUIDE FOR FRENCH*. https://www.microsoft.com/en-us/windows/phones.

Minsky, Marvin L. 1968. « Minsky - Matter, Mind and Models. » *International Federation of Information Processing Congress* 1: 45-49. https://groups.csail.mit.edu/medg/people/doyle/gallery/minsky/mmm.html.

Modugno, F, et B a Myers. 1997. « Visual Programming in a Visual Shellâ"A Unified Approach. » *Jvlc* 8: 491-522. doi:10.1006/jvlc.1997.0049.

Mollard, Yoan, Thibaut Munzer, Andrea Baisero, Marc Toussaint, et Manuel Lopes. 2015. « Robot Programming from Demonstration , Feedback and Transfer », 1825-31.

Moussa, F, C Kolski, et M Riahi. 2000. « A model based approach to semi-automated user interface generation for process control interactive applications. » *Interacting with Computers* 12 (3): 245-79. doi:10.1016/S0953-5438(99)00014-4.

MOUSSA, Faouzi. 2005. *Vers une méthodologie globale de conception et de génération semi-automatique des IHM pour les systèmes industriels*. Tunis, Tunisie.

Müller, a. 2009. « VDM—The Vienna Development Method. » *Bachelor thesis in" Formal Methods in Software ….* http://www.risc.uni-linz.ac.at/publications/download/risc_3820/main.pdf.

Myers, Brad a, et Richard Mcdaniel. 2000. « Demonstrational Interfaces: Sometimes You Need a Little Intelligence, Sometimes You Need a Lot », 45-60.

Myers, Brad A. 1998. « Scripting Graphical Applications by Demonstration. » In *Human Factors in Computing Systems (CHI'98)*, 534-41. Los Angeles, Californie: ACM/SIGCHI. http://www.cs.cmu.edu/~bam.

Myers, Bras A., et William Buxton. 1986. « Creating Highly-Interactive and Graphical User Interfaces by Demonstration » 20 (4): 1-21.

Nielsen, Jakob, et Rolf Molich. 1990a. *Heuristic evaluation of user interfaces*. *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*. New York, New York, USA: ACM Press. doi:10.1145/97243.97281.

———. 1990b. « Heuristic evaluation of user interfaces. » In *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, 249-56. New York, New York, USA: ACM Press. doi:10.1145/97243.97281.

Normand, V. 1992. « Le modèle SIROCO: de la spécification conceptuelle des interfaces utilisateur à leur réalisation. » Joseph-Fourier - Grenoble I.

Oliveira, Raquel, Sophie Dupuy-chessa, et Gaëlle Calvary. 2015. « Plasticity of User Interfaces : Formal Verification of Consistency. » In *7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 260-65. ACM.

OMG. 2003. *MDA Guide version 1.0.1*.

Oney, Stephen, et Brad Myers. 2009. « FireCrystal: Understanding interactive behaviors in dynamic web pages. » *2009 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2009*, 105-8. doi:10.1109/VLHCC.2009.5295287.

OTAN. 1968. « Conférences célèbres du comité scientifique de l'O.T.A.N à Garmisch. » In . en Allemagne 7 au 11 Octobre 1968, et à Rome du 27 au 31 Octobre 1969.

Palanque, Philippe, Rémi Bastide, et Marco Winckler. 2003. « Automatic generation of interactive systems: why a task model is not enough | Remi Bastide - Academia.edu. » In *Human-Computer Interaction: Theory and Practice*, 198-202. https://books.google.fr/books?id=s6YW-cZBTqwC&pg=PA198&lpg=PA198&dq=Automatic+generation+of+interactive+systems:+why+a+task+model+is+not+enough&source=bl&ots=TY8wcLQSUc&sig=MPQR5ZzDIzonWT-ey8-Y4o-4q8I&hl=fr&sa=X&ved=0ahUKEwiYkZCPz8rOAhXEVRoKHeUwDFgQ6AEIKT.

Pandey, Tulika, et Saurabh Srivastava. 2015. « Comparative Analysis of Formal Specification Languages Z, VDM and B. » *International Journal of Current Engineering and Technology E-ISSN* 5 (3): 2277-4106.

Panjaitan, Seno, et Georg Frey. 2006. « Combination of UML modeling and the IEC 61499 function block concept for the development of distributed automation systems. » *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 766-73. doi:10.1109/ETFA.2006.355405.

Paternò, F. 2012. *Model-based design and evaluation of interactive applications*. Springer Science & Business Media.

Paternò, F., C. Mancini, et S. Meniconi. 1997. « ConcurTaskTrees: A diagrammatic notation for specifying task models. » *Interact'97*, 362-69. doi:10.1.1.86.585.

Paterno, Fabio, Ilse Breedvelt-schouten M., et N M De Koning. 1999. « Deriving Presentations from Task Models. »

Paternò, Fabio, et Giorgio P. Faconti. 1992. « On the LOTOS use to describe graphical interaction. » In , 155-73. Cambridge University Press.

Pfaff, Günther E. 1983. « User Interface Management Systems. » In *the Workshop on User Interface Management Systems*. Seeheim: Springer. https://books.google.fr/books?hl=fr&lr=&id=RN-pCAAAQBAJ&oi=fnd&pg=PA3&dq=User+Interface+Management+Systems+Pfaff+1985&ots=9fGTii5J8j&sig=4N2ykWu7-UFBnt06qz6FQE3FP_w#v=onepage&q=User Interface Management Systems Pfaff 1985&f=false.

Pham, Hoang. 2006. *System software reliability*. Springer. doi:10.1007/1-84628-295-0.

Piernot, Philippe P., et Marc P. Yvon. 1995. « A Model for Incremental Construction of Command Trees. » In *HCI'95*, 169-79. Huddersfield, England. https://books.google.fr/books?hl=fr&lr=&id=OlZONLkAtVMC&oi=fnd&pg=PA169&dq=A+Model+for+Incremental+Construction+of+Command+Trees.&ots=cK8Rbxa4mM&sig=ylCQAZQZIiemLbAdsBk-7fZsiYY#v=onepage&q=A Model for Incremental Construction of Command Trees.&f=false.

Potier, Jean-Claude. 1995. « Conception sur exemple, mise au point et génération de programmes portables de géométrie paramétrée dans le système EBP. » *LISI/ENSMA*. Université de Poitiers.

Prat, Sophie, Philippe Rauffet, Pascal Berruet, et Alain Bignon. 2016. « A multi-level requirements modeling for sociotechnical system simulation-based checking A multi-level requirements modeling for sociotechnical system simulation-based checking », nº October.

Prat, Sophie, Philippe Rauffet, Alain Bignon, et Pascal Berruet. 2015. « Vers l'intégration d'une approche de génération automatique de modele de simulation dans un flot de conception de contrôle -commande Génération automatique de modèles de simulation dans un flot de conception : approche et outils », nº October.

Qt. 2016. « Qt | Cross-platform software development for embedded & desktop. » https://www.qt.io/.

Radeke, F, P Forbrig, A Seffah, D Sinnig, K Coninx, K Luyten, et K a Schneider. 2006. « PIM tool: Support for pattern-driven and model-based UI development. » *Task Models and Diagrams for Users Interface Design*, 82-96.

Radeke, Frank, et Peter Forbrig. 2007. « Patterns in Task-Based Modeling of User Interfaces. » *Task Models and Diagrams for User Interface Design* 5963: 184-97. doi:10.1007/978-3-540-77222-4_15.

Ramón, Óscar Sánchez, Jesús Sánchez Cuadrado, Jesús García Molina, et Jean Vanderdonckt. 2015. « A layout inference algorithm for graphical user interfaces. » *Information and Software Technology* 70: 155-75. doi:10.1016/j.infsof.2015.10.005.

Raneburger, David, Roman Popp, et Jean Vanderdonckt. 2012. « An automated layout approach for model-driven WIMP-UI generation. » *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12*, 91. doi:10.1145/2305484.2305501.

Rauffet, Philippe, Gael Morel, Pascal Berruet, et Christine Chauvin. 2013. « Approche pluridisciplinaire pour la conception des systèmes sociotechniques. » *Journal national de la recherche en IUT* 4 (October): 1-12.

Ravichandran, B Y T, et Marcus a Rothenberger. 2003. « Component Markets. » *ACM* 46 (8).

Rechard, Julien. 2015. « Introduction de critères ergonomiques dans un système de gération automatique d ' interface de supervision. » Université de Bretagne Sud.

Rechard, Julien, Alain Bignon, Pascal Berruet, et Thierry Morineau. 2015. « Verification and validation of a Work Domain Analysis with turing machine task analysis. » *Applied Ergonomics* 47. Elsevier Ltd: 265-73. doi:10.1016/j.apergo.2014.10.012.

Richard Turton, Richard C. Bailie, Wallace B. Whiting, Joseph a. Shaelwitz. 2008. *Analysis, Synthesis and Design of Chemical Processes Third Edition*. *Journal of Chemical Information and Modeling*. Vol. 53. doi:10.1017/CBO9781107415324.004.

Rothermel, G., L. Li, C. DuPuis, et M. Burnett. 1998. « What you see is what you test: a methodology for testing form-based visual programs. » *Proceedings of the 20th International Conference on Software Engineering*, 198-207. doi:10.1109/ICSE.1998.671118.

Rothermel, Gregg, Margaret Burnett, Lixin Li, Christopher Dupuis, et Andrei Sheretov. 2001. « A methodology for testing spreadsheets. » *ACM Transactions on Software Engineering and Methodology* 10 (1): 110-47. doi:10.1145/366378.366385.

Rugaber, Spencer, et Kurt Stirewalt. 2004. « Model-Driven Reverse Engineering. » *IEEE software* 21 (4): 45-53.

Ruvini, Jean-David, et Christophe Dony. 2001. « Learning Users' Habits to Automate Repetitive Tasks. » *Lierberman, H Your Wish Is My Command: Programming by Example. Morgan Kaufmann, Sao Francisco CA*, 271-95.

Samaan, Kinan. 2006. « Prise en Compte du Modèle d'Interaction dans le Processus de Construction et d'Adaptation d'Applications Interactives. » *Laboratoire Interaction Collaborative Téléformation Téléactivités (ICTT)*, 218. http://kinan.samaan.free.fr/.

Sannella, Donald, et Andrzej Tarlecki. 1988. « Toward formal development of programs from algebraic specifications: Implementations revisited. » *Acta Informatica* 25 (3). Springer Berlin Heidelberg: 233-81. doi:10.1007/3-540-17660-8_50.

Sannella, Donald, et Andrzej Torlecki. 1985. « PROGRAM SPEClFICATION AND DIWELOPMEN ' I IN STANDARD ML. » *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 67-77.

Sanou, Loé. 2008. « Définition et réalisation d'une boîte à outils générique dédiée à la Programmation sur Exemple. » Université de Poitiers.

Scaffidi, Christopher. 2010. « Sharing, finding and reusing end-user code for reformatting and validating data. » *Journal of Visual Languages and Computing* 21 (4). Elsevier: 230-45. doi:10.1016/j.jvlc.2010.06.001.

Seffah, Ahmed. 2015. *Patterns of HCI Design and HCI Design of Patterns: Bridging HCI Design and Model-Driven Software Engineering*. Édité par Desney Tan et Jean Vanderdonckt. *Media*. Human-Comp. Springer.

Segal, Judith. 2007. « Design planning in end-user Web development. » *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2007*, 189-96. doi:10.1109/VLHCC.2007.17.

Senach, Bernard. 1990. « Evaluation ergonomique des interfaces homme-machine : une revue de la littérature. » INRIA.

Singh, Raghu. 1996. « International Standard ISO/IEC 12207 Software Life Cycle Processes. » *Software Process: Improvement and Practice* 2: 35-50. doi:10.1002/(SICI)1099-1670(199603)2:1<35::AID-SPIP29>3.0.CO;2-3.

Sinnig, Daniel. 2004. « The complicity of patterns and model-based UI development. » CONCORDIA, Montreal, Quebec, CANADA.

Smith, David C. 1977. *A Computer Program to Model and Stimulate Creative Thought*. Basel: Birkhauser.

Sottet, Jean-Sebastien, Gaëlle Calvary, Jean-Marie Favre, et Joëlle Coutaz. 2006. « IHM & IDM : Un tandem prometteur. » In *Ergo'IA*.

Spivey, J.M. 1989. « An introduction to Z and Formal Specifications. » *Software Engineering Journal* 4: 40-50. doi:10.1049/sej.1989.0006.

Standish Group. 1995. « CHAOS Report. » *Group*, 11-13. doi:0674.

———. 2009. « CHAOS Summary 2009. » *The Standish Group*, 1-4.

———. 2014. « The Standish group: the chaos report. » *Project Smart*, 16. doi:10.1016/S0895-7061(01)01532-1.

Subrahmaniyan, Neeraja, Cory Kissinger, Kyle Rector, Derek Inman, Jared Kaplan, Laura Beckwith, et Magaret Burnett. 2007. « Enhancing the programmability of spreadsheets with logic programming. » *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2007*, 87-94. doi:10.1109/VLHCC.2007.18.

Szekely, Pedro. 1996. « Retrospective and Challenges for Model-Based Interface Development. » *Design, Specification and Verification of Interactive Systems '96*, 1-27. doi:10.1007/978-3-7091-7491-3_1.

Texier, Guillaume. 2000. « Contribution à l'ingénierie des systèmes interactifs : Un environnement de conception graphique d'applications spécialisées de conception. » ENSMA/ Université de Poitiers.

Thramboulidis, K. 2012. « IEC 61499 : Back to the well proven practice of IEC 61131 ? » In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, 1-8. IEEE.

To, Combemale. 2008. « Benoˆıt Combemale To cite this version »: UNIVERSITÉ DE TOULOUSE.

Trætteberg, Hallvard. 2002. « Model-based User Interface Design. » *Science And Technology*. Faculty of Information Technology, Mathematics and Electrical Engineering Norwegian University of Science and Technology.

Tran, Vi, Jean Vanderdonckt, Manuel Kolp, et Stéphane Faulkner. 2009. « Generating user interface from task, user and domain models. » *2nd International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies, and Services - CENTRIC 2009*, 19-26. doi:10.1109/CENTRIC.2009.24.

Trudel, Sylvie. 2012. « USING THE COSMIC FUNCTIONAL SIZE MEASUREMENT METHOD (ISO 19761) AS A SOFTWARE REQUIREMENTS IMPROVEMENT MECHANISM. » ÉCOLE DE TECHNOLOGIE SUPÉRIEURE UNIVERSITÉ DU QUÉBEC.

Van Deursen, Arie, et Paul Klint. 2002. « Domain-Specific Language Design Requires Feature Descriptions. » *Journal of Computing and Information Technology* 10: 1-17. doi:10.2498/cit.2002.01.01.

Vanderdonckt, Jean. 1995. « Knowledge-based systems for automated user interface generation: the TRIDENT experience. » *Proceedings of the CHI*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.8207&rep=rep1&type=pdf.

Vanderdonckt, Jean, et Xavier Gillo. 1994. « Visual techniques for traditional and multimedia layouts. » *Proceedings of the workshop on Advanced visual interfaces*, 95-104. doi:http://doi.acm.org/10.1145/192309.192334.

VEPSÄLÄINEN, Timo, David HÄSTBACKA, et Seppo KUIKKA. 2008. « ool support for the UML automation profile-for domain-specific software development in manufacturing. » In *Software Engineering Advances, 2008. ICSEA'08. The Third International Conference*, 43-50. IEEE.

Vernes, Picardie Jules. 2008. « Algorithmes de Contrôles Avancés pour les Installations à Gaz du LHC au CERN suivant le Framework et l ' approche dirigée par les modèles du projet GCS Docteur de l ' Université de Picardie Jules Vernes. »

Vicente, K. J. 1999. *Cognitive work analysis: Toward safe, productive, and healthy computer-based work*. CRC Press.

Vyatkin, Valeriy. 2009. « The IEC 61499 standard and its semantics - Bridging the Gap Between PLC Programming Languages and Distributed Systems. » *Industrial Electronics Magazine, IEEE* 3: 40-48. doi:10.1109/MIE.2009.934796.

Wiegers, Karl, et Joy Beatty. 2013. *Software Requirements*. Édité par Devon Musgrave. *Managing*. 3rd éd. Microsoft Press.

Wirsing, M. 1995. « Algebraic specification languages: An overview. » *In Recent Trends in Data Type Specification. Springer Berlin Heidelberg*, 81-115.

Wolff, A., P. Forbrig, A. Dittmar, et D. Reichart. 2005. « Linking GUI Elements to Tasks - Supporting an Evolutionary Design Process. » In *4th international workshop on Task models and diagrams*, 27-34. ACM.

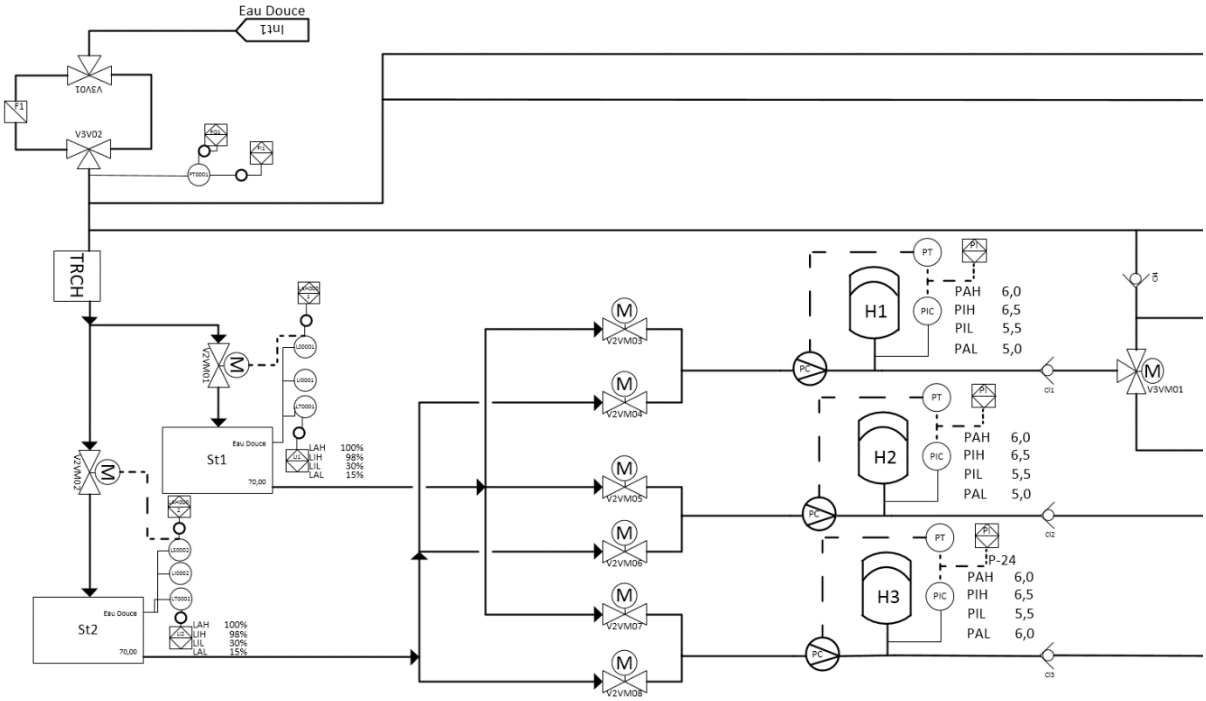
Wolff, Andreas, et Peter Forbrig. 2009. « Deriving User Interfaces from Task Models. » In *MDDAUI'09*, 439:4. Sanibel Island, USA: CEUR-WS. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-439/.

Wurdel, M, D Sinnig, et P Forbrig. 2008. « {CTML}: Domain and Task Modeling for Collaborative Environments. » *Journal of Universal Computer Science* 14 (19): 3188-3201. doi:http://dx.doi.org/10.3217/jucs-014-19-3188.

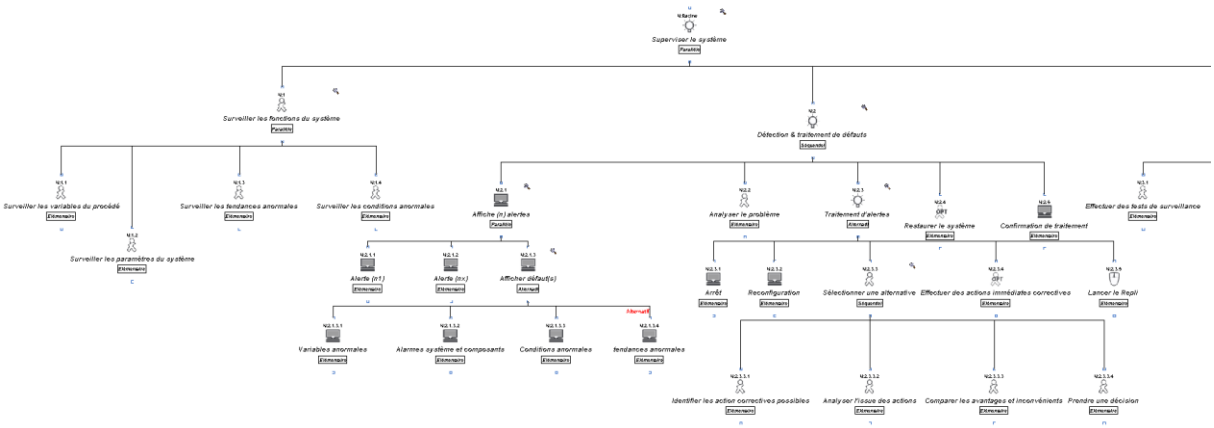
Zaki, Michael, et Peter Forbrig. 2012. « Towards the Generation of Assistive User Interfaces for Smart Meeting Rooms Based on Activity Patterns. » *Ambient Intelligence*, 288-95. doi:10.1007/978-3-642-34898-3_19.

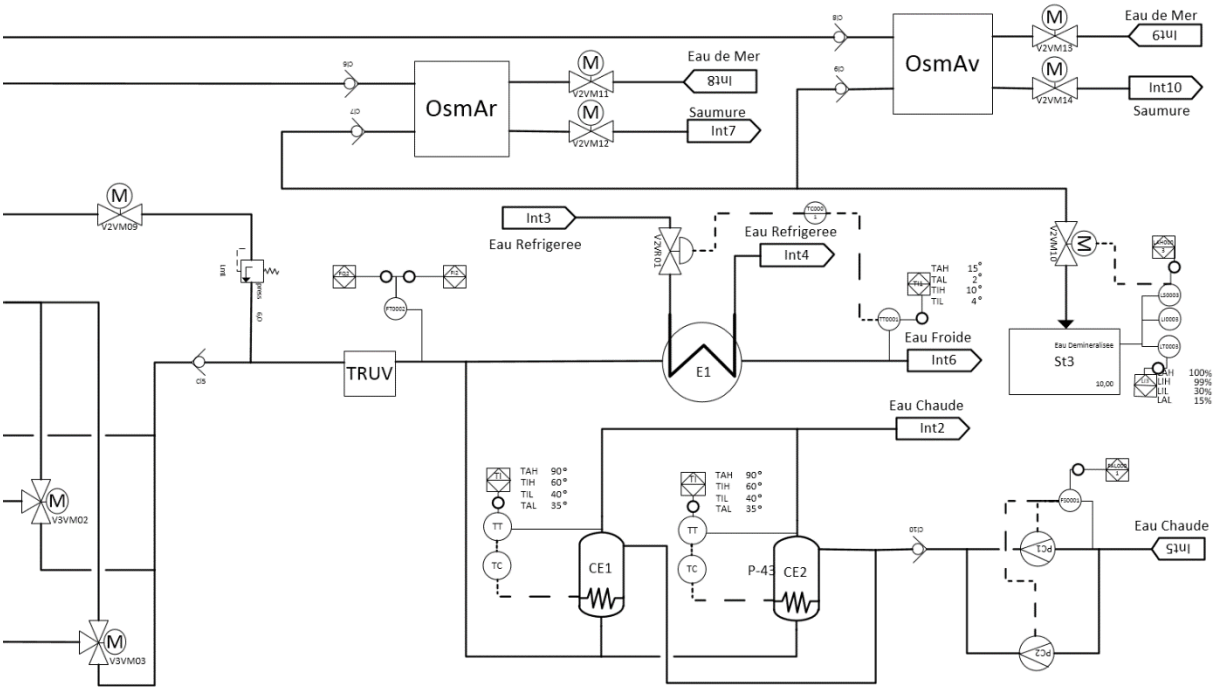
Zhou, Michelle X., et Steven K. Feiner. 1997. « Top-down hierarchical planning of coherent visual discourse. » *Proceedings of the 2nd international conference on Intelligent user interfaces - IUI '97* 97: 129-36. doi:10.1145/238218.238314.

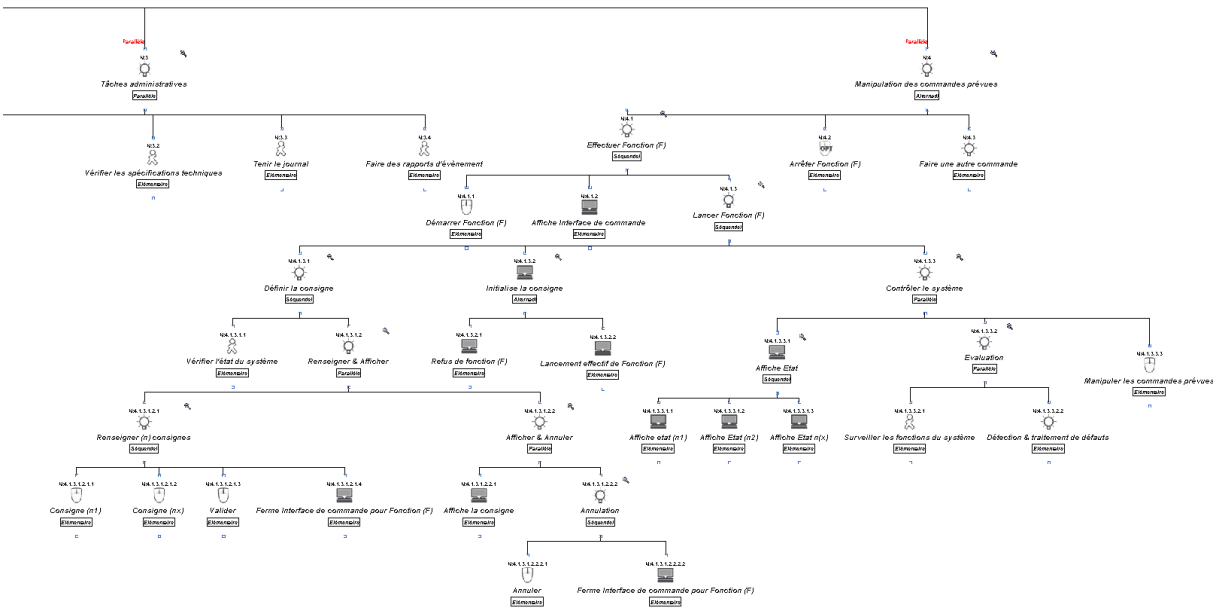
1. P&ID du système EdS (Eau douce Sanitaire)



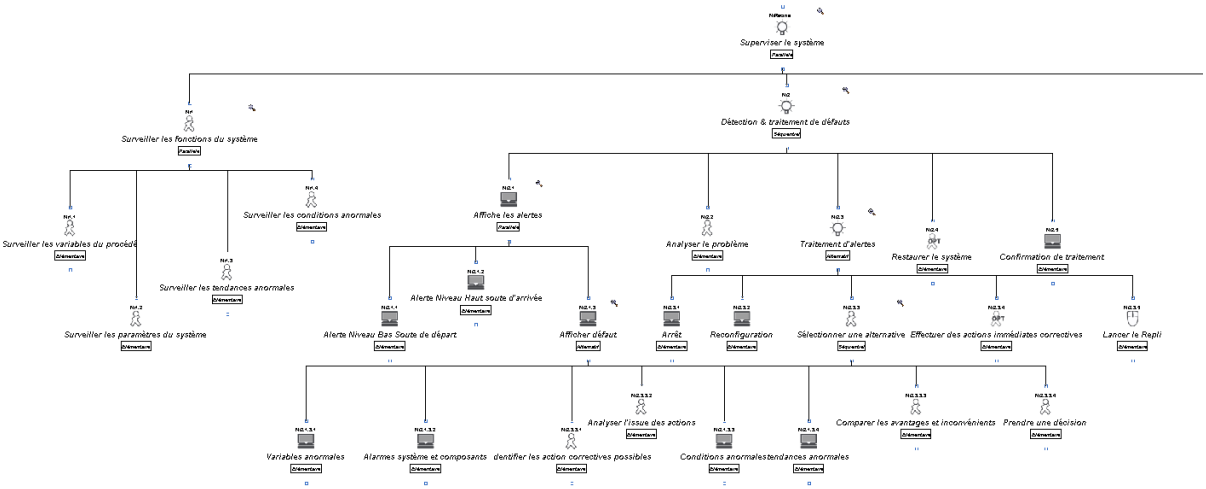
2. Pattern de tâches

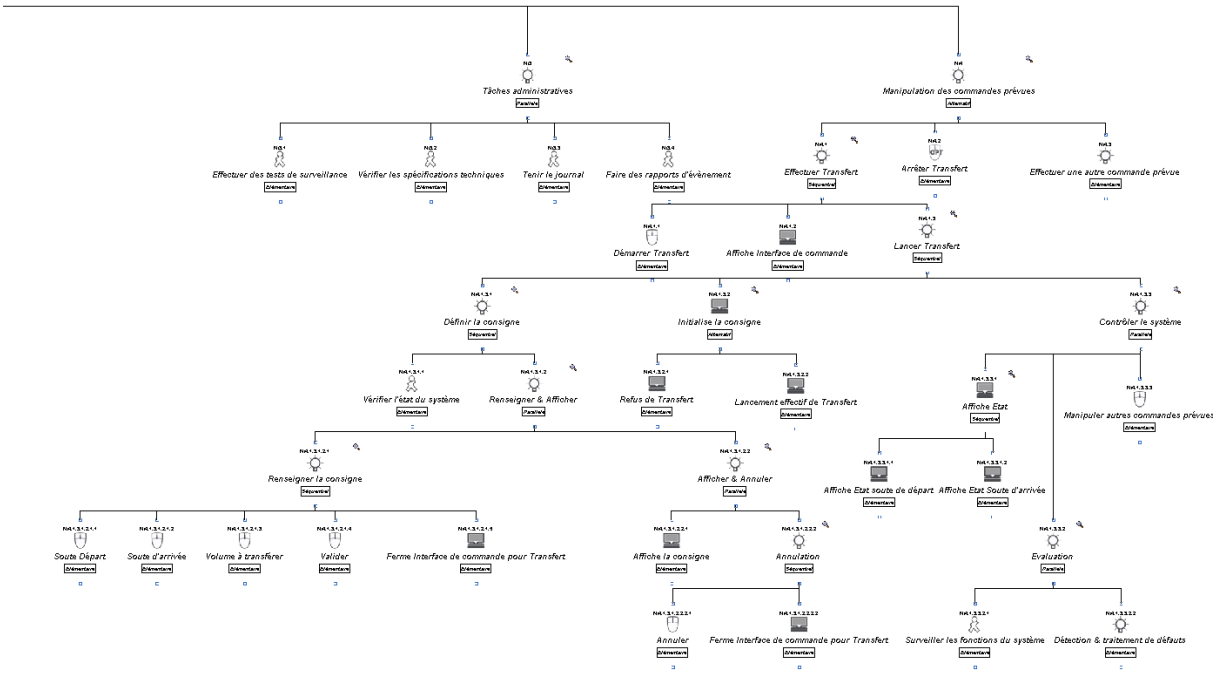



OsmAr
OsmAv
Eau de Mer
Int18
Saumure
Int7
Eau de Mer
Int19
Int10
Saumure
V2VM09
V2VM11
V2VM12
V2VM13
V2VM14
V2VM01
V2VM10
Int3
Eau Refrigeree
Eau Refrigeree
Int4
Eau Froide
Int6
TRUV
E1
St3
Eau Chaude
Int2
Eau Chaude
Int5
V3VM02
V3VM03
CE1
CE2
P-43
TAH 90°
TIH 60°
TIL 40°
TAL 35°


Tâches administratives
Manipulation des commandes prévues
Vérifier les spécifications techniques
Tenir le journal
Faire des rapports d'évènement
Effectuer Fonction (F)
Arrêter Fonction (F)
Faire une autre commande
Démarrer Fonction (F)
Affiche Interface de commande
Lancer Fonction (F)
Définir la consigne
Initialise la consigne
Contrôler le système
Vérifier l'état du système
Renseigner & Afficher
Refus de fonction (F)
Lancement effectif de Fonction (F)
Affiche Etat
Evaluation
Manipuler les commandes prévues
Renseigner (n) consignes
Afficher & Annuler
Affiche etat (n1)
Affiche Etat (n2)
Affiche Etat n(x)
Surveiller les fonctions du système
Détection & traitement de défauts
Consigne (n1)
Consigne (nx)
Valider
Ferme Interface de commande pour Fonction (F)
Affiche la consigne
Annulation
Annuler
Ferme Interface de commande pour Fonction (F)

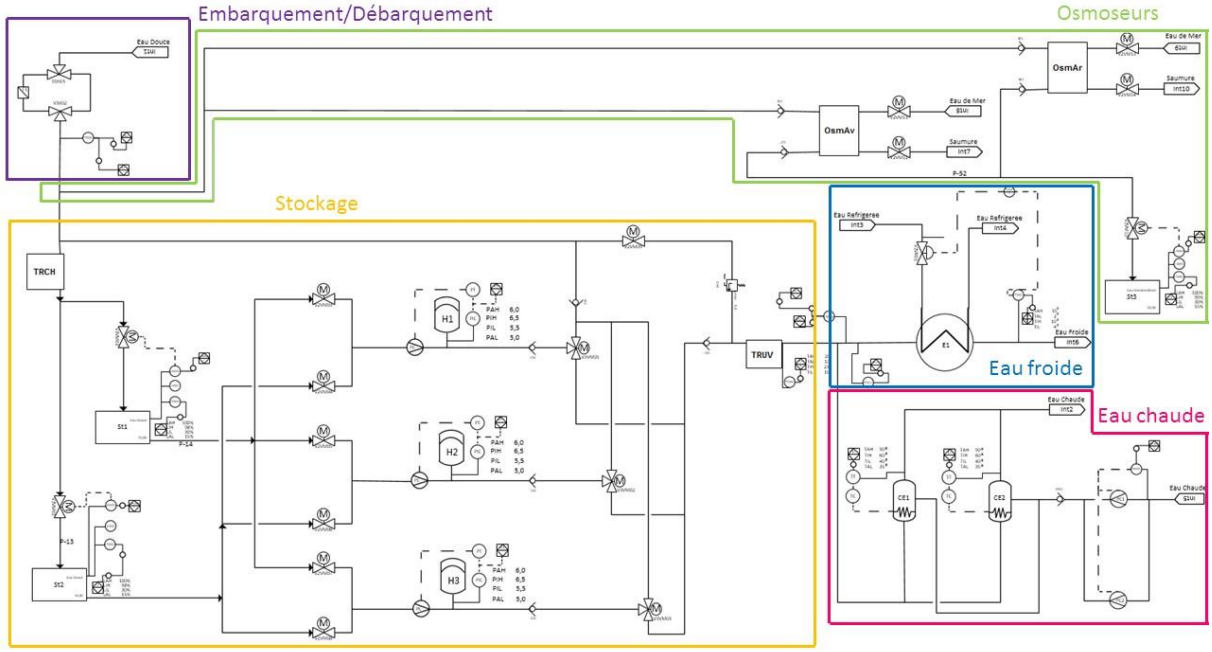
3. Modèle de tâches de la fonction de transfert




Tâches administratives
Manipulation des commandes prévues
Effectuer des tests de surveillance
Vérifier les spécifications techniques
Tenir le journal
Faire des rapports d'évènement
Effectuer Transfert
Arrêter Transfert
Effectuer une autre commande prévue
Démarrer Transfert
Affiche Interface de commande
Lancer Transfert
Définir la consigne
Initialise la consigne
Contrôler le système
Vérifier l'état du système
Renseigner & Afficher
Refus de Transfert
Lancement effectif de Transfert
Affiche Etat
Manipuler autres commandes prévues
Renseigner la consigne
Afficher & Annuler
Affiche Etat soute de départ
Affiche Etat Soute d'arrivée
Soute Départ
Soute d'arrivée
Volume à transférer
Valider
Ferme Interface de commande pour Transfert
Affiche la consigne
Annulation
Evaluation
Annuler
Ferme Interface de commande pour Transfert
Surveiller les fonctions du système
Détection & traitement de défauts

4. Grille d'évaluation de la compréhension du schéma PID

Découpage du schéma en sous-zones :



Zone embarquement/débarquement

Compréhension du fonctionnement des éléments	
Critère	**Cocher :**
Le participant connaît/comprend le fonctionnement de tous les éléments.	☐
Le participant connaît/comprend le fonctionnement de certains éléments.	☐
Le participant ne connaît pas les éléments.	☐
Compréhension de l'agencement du circuit	
Critère	**Cocher :**
Le participant comprend correctement l'agencement du circuit.	☐
Le participant comprend partiellement l'agencement du circuit.	☐
Le participant ne comprend pas l'agencement du circuit.	☐

Zone osmoseurs

Compréhension du fonctionnement des éléments	
Critère	**Cocher :**
Le participant connaît/comprend le fonctionnement de tous les éléments.	☐
Le participant connaît/comprend le fonctionnement de certains éléments.	☐
Le participant ne connaît pas les éléments.	☐
Compréhension de l'agencement du circuit	
Critère	**Cocher :**
Le participant comprend correctement l'agencement du circuit.	☐
Le participant comprend partiellement l'agencement du circuit.	☐
Le participant ne comprend pas l'agencement du circuit.	☐

Zone stockage

Compréhension du fonctionnement des éléments	
Critère	**Cocher :**
Le participant connaît/comprend le fonctionnement de tous les éléments.	☐
Le participant connaît/comprend le fonctionnement de certains éléments.	☐
Le participant ne connaît pas les éléments.	☐
Compréhension de l'agencement du circuit	
Critère	**Cocher :**
Le participant comprend correctement l'agencement du circuit.	☐
Le participant comprend partiellement l'agencement du circuit.	☐
Le participant ne comprend pas l'agencement du circuit.	☐

Zone eau froide

Compréhension du fonctionnement des éléments	
Critère	**Cocher :**
Le participant connaît/comprend le fonctionnement de tous les éléments.	☐
Le participant connaît/comprend le fonctionnement de certains éléments.	☐
Le participant ne connaît pas les éléments.	☐
Compréhension de l'agencement du circuit	
Critère	**Cocher :**
Le participant comprend correctement l'agencement du circuit.	☐
Le participant comprend partiellement l'agencement du circuit.	☐
Le participant ne comprend pas l'agencement du circuit.	☐

Zone eau chaude

Compréhension du fonctionnement des éléments	
Critère	**Cocher :**
Le participant connaît/comprend le fonctionnement de tous les éléments.	☐
Le participant connaît/comprend le fonctionnement de certains éléments.	☐
Le participant ne connaît pas les éléments.	☐
Compréhension de l'agencement du circuit	
Critère	**Cocher :**
Le participant comprend correctement l'agencement du circuit.	☐
Le participant comprend partiellement l'agencement du circuit.	☐
Le participant ne comprend pas l'agencement du circuit.	☐

☐ Le participant comprend le fonctionnement du schéma à partir des explications de l'expérimentateur.

5. Mémo

MEMO pour tests utilisateurs

L'interface de spécification se compose d'un **enregistreur** qui permet d'enregistrer les actions de l'opérateur pour réaliser une fonction et d'un **rejoueur** qui permet de simuler, vérifier et valider les fonctions enregistrées.

ENREGISTREMENT D'UNE FONCTION *(EXEMPLE AVEC LA FONCTION BRASSAGE)*

Etape 1 : Sélection de la fonction.	Pour commencer l'enregistrement, sélectionnez la fonction voulue dans la liste *(ici, « brassage »)*.
Etape 2 : Sélection des points de départs et d'arrivée possibles.	1) Choisissez les différents points de départ possibles du brassage *(soute 1 ou soute 2). Sélectionnez l'opérateur « ou ».* 2) Faites de même pour la sélection des points d'arrivée.
Etape 3 : Choix de tous les points de passage possibles.	1) Choisissez les hydrophores *(H1, H2 & H3)* et sélectionnez l'opérateur « ou ». 2) Cochez tous les chemins possibles parmi la liste proposée. *Pour le brassage, tous les chemins proposés le sont.*
Etape 4 : Configuration du circuit.	1) Sélectionnez une soute de départ *(soute 1)* puis un point d'arrivée *(soute 1)*. 2) Ouvrez les vannes qui permettent de réaliser le brassage entre ces deux soutes *(p.ex. V2VM01, V2VM03 & V3VM01)*, puis la pompe *(p.ex. H1)*.
Etape 5 : Renseignement des conditions d'arrêt.	1) Indiquez la ou les conditions d'arrêt de la fonction. *Ici, le volume de la soute d'arrivée doit être égal au volume du compteur.* 2) Choisissez si vous souhaitez donner la possibilité ou non à l'opérateur d'arrêter manuellement *le brassage. Il est préférable de laisser la possibilité à l'opérateur d'arrêter le brassage.*
Etape 6 : Fermeture de tous les éléments ouverts sur le circuit	1) Arrêtez la pompe. 2) Fermez ensuite les vannes.
Etape 7 : Choix d'une deuxième configuration.	1) Renouvelez les 3 étapes précédentes (étapes 4 à 6) avec une configuration différente, *en demandant cette fois un brassage de la soute 2 à la soute 2.* 2) A la fin de l'étape 5, choisissez la priorité que vous souhaitez accorder à cette fonction de manière générale.

REJEU D'UNE FONCTION

1) Rendez-vous dans le « rejoueur ».

2) Sélectionnez la fonction souhaitée.

3) Un bouton apparaît, portant le nom de la fonction, c'est celui qui apparaitra sur l'IHM finale. Cliquez dessus. Un bandeau apparaît, il permettra à l'opérateur de renseigner la consigne de lancement de la fonction lorsqu'il sera sur l'IHM finale.

4) Renseignez la consigne comme indiqué et la réalisation de la fonction sera simulée selon ce que vous avez demandé. Le système vous proposera de vérifier plusieurs configurations mais, pour ces tests, vous ne devrez en réaliser qu'une seule.

A vous de jouer ! Spécifiez les fonctions Transfert, Débarquement & Distribution

6. Questionnaire complémen²taire

Question	Commentaire libre
Quel est votre impression générale sur l'interface ?	
Quels sont les points positifs ?	
Quels aspects trouveriez-vous utiles d'améliorer ?	
Des suggestions ?	
Autres remarques	
Avez-vous eu à définir de telles spécifications lors de vos stages précédents ?	

7. Groupements des commentaires des participants à propos des qualités de l'interface

Groupement	Commentaires
Facile à utiliser (6 occurrences)	*« Facile d'utilisation » (Participant 1)* *« Facile à utiliser » (Participant 8)* *« Facilité d'utilisation » (Participant 11)* *« Facile à utiliser » (Participant 12)* *« Facile d'utilisation » (Participant 13)* *« Facile à utiliser » (Participant 15)*
Intuitif (7 occurrences)	*« Intuitif à utiliser » (Participant 4)* *« Utilisation assez intuitive » (Participant 6)* *« Extrêmement intuitif » (Participant 7)* *« Symboles des différents éléments plutôt instinctifs à comprendre » (Participant 10)* *« Assez intuitif » (Participant 11)* *« Globalement intuitif » (Participant 12)* *« Assez intuitif » (Participant 16)*
Simple (8 occurrences)	*« L'utilisation du logiciel m'a semblée très simple » (Participant 3)* *« Assez simple » (Participant 4)* *« Simple d'utilisation » (Participant 5)* *« L'interface est simple, épurée » (Participant 9)* *« Utilisation simple » (Participant 10)* *« Simple d'utilisation » (Participant 10)* *« Menu simple et clair ET efficace » (Participant 14)* *« Simple d'utilisation » (Participant 16)*
Clair (6 occurrences)	*« Interface claire, très bien expliquée » (Participant 3)* *« Clair » (Participant 8)* *« Plutôt clair » (Participant 8)* *« Claire » (Participant 16)* *« Menu simple et clair ET efficace » (Participant 14)* *« Moyens d'action clairs et bien définis » (Participant 14)*
Utile (3 occurrences)	*« Utile » (Participant 13)* *« Répond à une demande » (Participant 8)* *« Outil qui pourra se révéler très utile » (Participant 12)*
Utilisation guidée (4 occurrences)	*« Bien indiquée/renseignée » (Participant 1)* *« Utilisation guidée » (Participant 2)* *« L'interface guide l'utilisation » (Participant 11)* *« L'utilisateur sait directement ce qui est important » (Participant 12)*
Agréable (2 occurrences)	*« Agréable » (Participant 1)* *« Interface agréable » (Participant 2)*
Ludique (2 occurrences)	*« Ludique » (Participant 1)* *« Ludique » (Participant 15)*
Pratique (2 occurrences)	*« Très pratique » (Participant 11)* *« Pratique » (Participant 13)*
Epurée (3 occurrences)	*« Interface très lisible et épurée » (Participant 7)* *« Design épuré » (Participant 7)* *« Pas de surcharge d'éléments ou d'information » (Participant 12)*
Autres :	*« Bonne impression » (Participant 2)* *« Ergonomique » (Participant 2)* *« Sobre » (Participant 12)* *« Adaptée à tous » (Participant 15)* *« Interface compréhensible » (Participant 15)*

Résumé

Pour la conception d'un système contrôle-commande, les spécifications fonctionnelles sont à la charge des concepteurs car ce sont eux qui en maîtrisent le fonctionnement. Ces experts n'ont pourtant généralement pas les connaissances en programmation de ceux qui conçoivent le système de pilotage. Ils écrivent alors ces spécifications fonctionnelles en langage naturel, les communiquent ensuite aux concepteurs de l'interface de supervision et du programme de commande qui sont en charge de les implémenter et de les intégrer au système. Les erreurs qui découlent de l'interprétation des spécifications émanent de la différence de culture technique entre les différents intervenants du projet. De plus, suivant la complexité du système, la définition des spécifications fonctionnelles peut être fastidieuse.

Nous proposons une approche de conception basée d'une part sur l'analyse de la tâche et sur les techniques du *End User Development* pour l'obtention de spécifications fonctionnelles validées par les experts métiers.

Les techniques de l'ingénierie dirigée par les modèles sont mises en œuvre pour générer automatiquement l'interface de spécification (qui intègre un *Enregistreur,* un *Généralisateur,* un *Rejoueur,* et un *Correcteur*), l'interface de supervision du système à piloter et son programme de commande.

La démarche proposée a fait l'objet d'une preuve de concept démontrant sa faisabilité technique. Cette preuve de concept a fait l'objet d'évaluations qui ont démontré son intérêt dans le cadre de la conception de système de supervision.

Mots-clés : Spécification fonctionnelle, End User Development (EUD), Modèles de tâches, Interaction Homme-Machine (IHM), Ingénierie dirigée par les modèles (IDM), Supervision industrielle, Contrôle-commande.

Abstract

For designing complex and sociotechnical systems, business experts are responsible for writing the functional specifications because of their operational expert knowledge. However, these experts do not usually own the programming knowledge of those who design supervision systems. The task of the system design expert is then to define the functional specifications. S/he writes them in natural language, and then provides them to the designers of the supervision interface and the control-command code. The designers' job is then to implement and integrate the specifications into the system. Errors from the specification interpretation come from the difference of technical knowledge between the various partners involved in the project. Moreover, depending on the complexity of the system, the definition of functional specifications can be tedious.

We propose a design approach based on task modelling and End User Development in order to obtain functional specifications validated by the business experts (mechanical engineer for example).

Model-driven engineering techniques are implemented to automatically generate the specification interface (that integrates Recorder, Generalizer, Replayer, and Corrector), the system supervision interface to be piloted and its control program.

The technical feasibility of the proposed approach was demonstrated through a proof of concept. This proof of concept was evaluated to demonstrate the interest of the approach in the design of supervision systems.

Keywords : Functional Specification, End User Development, Task analysis, Human Computer Interaction (HCI), Model Driven engineering (MDE), Industrial supervision, Control-command.