Intégration de techniques de vérification par simulation dans un processus de conception automatisée de contrôle commande

Sophie Prat

THESE / UNIVERSITE DE BRETAGNE SUD
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de :
DOCTEUR DE L'UNIVERSITE DE BRETAGNE-SUD

Mention : STIC
Ecole Doctorale : SICMA

Présentée par

Sophie PRAT

Préparée à l'unité mixte de recherche :
Lab-STICC UMR 6285

Etablissement de rattachement :
Université de Bretagne Sud

Intégration de Techniques de Vérification par Simulation dans un Processus de Conception Automatisée de contrôle-commande

Thèse soutenue le 6 décembre 2017,
devant le jury composé de :

M. Bernard RIERA
Professeur, Université de Reims Champagne-Ardenne / Rapporteur

M. Thierry SORIANO
Professeur, Université de Toulon / Rapporteur

M. Pierre CASTAGNA
Professeur, Université de Nantes / Examinateur

Mme. Pascale CHIRON
Maître de Conférences, Ecole Nationale d'Ingénieurs de Tarbes / Examinateur

Mme. Pauline RIBOT
Maître de Conférences, Université Toulouse III Paul Sabatier / Examinateur

M. Pascal BERRUET
Professeur, Université de Bretagne Sud / Directeur de thèse

M. Philippe RAUFFET
Maître de Conférences, Université de Bretagne Sud / Co-encadrant de thèse

M. Alain BIGNON
Docteur, SEGULA Technologies / Co-encadrant de thèse

A Loulou,
(car il en faut de la patience !)

Remerciements

Tout d'abord il me faut remercier Michel COMBACAU sans qui cette aventure n'aurait jamais commencé. Puis, Alain BIGNON, Philippe RAUFFET et Pascal BERRUET, qui m'ont encadrée tout au long de ce parcours. Vous ne le savez peut-être pas, mais vous m'avez apporté beaucoup (que se soit par votre soutien, vos précieux conseils, ou tout simplement par votre bonne humeur et votre humour).

Je remercie également Bernard RIERA et Thierry SORIANO qui ont acceptés d'être rapporteur, ainsi que Pierre CASTAGNA, Pascale CHIRON et Pauline RIBOT qui ont acceptés d'examiner mes travaux.

Impossible de ne pas remercier mes collègues du projet Anaxagore : Djamal, Soraya, Olga, Laurianne, sans oublier le papa du projet (encore merci Alain), et les assimilés (Fabien je pense à toi) ! Je remercie également Jérémy, Florine et Marion qui furent de passage dans l'équipe. Merci également aux collègues (anciens et actuels) de SEGULA de Lorient pour les rigolades pendant les pauses, et/ou les discussions autour d'une bière(s). Merci à Seb (x2), Manu, Laurent, Johan, Eric, Davy, Renaud et les autres que j'oublie. Vincent merci pour les bières, et la découverte de la culture bretonne. Périne merci pour tout, et merci d'avoir pris le temps de relire quelques passages de ce manuscrit. Nico, un énorme merci pour les week-end "retour à la terre" permettant d'oublier la thèse quelques instants. Je remercie également les collègues de SEGULA Toulouse qui m'ont soutenue pendant ma phase terminale. Une pensée particulière à Gaëlle et Françoise qui ont veillé à ce que je me nourrisse le midi. Je tiens également à remercier Pierre GERVAIS, pour sa bienveillance.

Merci aussi aux collègues doctorant(e)s. Je remercie les anciens : Cédric, Samantha, Benoît, Julien, Tchin et Yu, et Alex qui a récemment rejoint le club des anciens. Un immense merci aux filles, oui Fanny et Amandine c'est bien de vous qu'il s'agit : vous êtes au top ! Merci Thomas pour les pauses clopes, et une petite pensée pour les nouveaux !

Je remercie également ma famille et les amis pour leur soutien (même si aucun d'entre eux ne comprend réellement ce que je fais). Linou, Tatane, Pascal (c'était chouette le camping-car) et Sarah merci d'être passés ! Merci Hélène pour tes encouragements, et merci Micka pour l'initiation au tricot : très efficace pour se vider la tête, et avoir chaud l'hiver !

En bref, merci à tout ceux (et celles) qui ont contribué de près ou de loin à cette aventure (et merci à la patience de IV) !

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

SEGULA Technologies
165 rue de la Montagne du Salut
BP 50256 - 56602 Lanester Cedex
www.segula.fr

Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) – UMR 6285
Université de Bretagne Sud
Centre de Recherche Christiaan Huygens
Rue de Saint Maudé 56321 Lorient Cedex
www.labsticc.fr

Table des matières

Table des figures

Liste des tableaux

Listes des publications

Conférence nationale avec comité de lecture

1) Prat, S., Rauffet, P., Bignon, A., & Berruet, P. (2015, June). Vers l'intégration d'une approche de génération automatique de modèle de simulation dans un flot de conception de contrôle-commande. In *JDJN MACS*.

Conférence internationale avec comité de lecture

2) Prat, S., Rauffet, P., Berruet, P., & Bignon, A. (2016, October). A multi-level requirements modeling for sociotechnical system simulation-based checking. In *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on* (pp. 001850-001855). IEEE.

3) Prat, S., Cavron, J., Kesraoui, D., Rauffet, P., Berruet, P., & Bignon, A. (2017, July). An automated generation approach of simulation moedels for checking control/monitoring system. In *20th IFAC World Congress*. Elsevier

Introduction générale

De nos jours, nous ne cherchons plus à concevoir des objets mais des systèmes, systèmes qui sont de plus en plus complexes. Il n'est plus question de concevoir des systèmes purement techniques, mais des systèmes sociotechniques où dimension technique et humaine sont en forte interaction. Ces systèmes sociotechniques peuvent être de tailles diverses, cela peut être un navire ou un avion comme un téléphone.

Le contexte économique a également évolué. Les industriels sont dans un état de concurrence généralisée, leur objectif n'est pas seulement de concevoir de nouveaux systèmes, il s'agit de le faire le plus vite possible et à moindre coût.

Face à l'évolution des systèmes à concevoir et à l'évolution du contexte économique, les méthodologies de conception évoluent. La nature des composants hétérogènes d'un système et leur forte intégration fait qu'il n'est plus possible de découper un système en autant de parties que de disciplines. Le système doit être conçu comme un tout, les équipes de conception devenant ainsi multi-disciplinaires. De plus, afin de réduire les temps de conception, il n'est plus question d'attendre qu'une étape du processus de conception soit finie pour démarrer la suivante. Les processus de conception ne sont donc plus séquentiels, ils deviennent concourants.

Afin d'éviter les coûts de re-conception, les industriels cherchent également à intégrer des processus de vérifications et de validations tout au long de la conception. L'intérêt est notamment d'éviter la détection tardive d'erreurs de conception, mais également de s'assurer de l'adéquation entre le système conçu et les exigences du client ainsi que celles de l'utilisateur final. Cependant, bien que visant à réduire les coûts de re-conception, la mise en oeuvre de boucles de validations et vérifications a un coût et peut potentiellement rallonger le temps de conception (puisqu'il est rajouté des étapes au processus de conception).

Ceci est d'autant plus vrai dans le cadre d'activités de supervision industrielle. Prenons comme exemple le cas de la conduite d'un navire. L'équipage supervise la conduite du navire au travers d'Interfaces Homme-Machine (IHM) qui permettent, outre la surveillance de l'état du navire, de pouvoir interagir avec ce dernier pour envoyer des commandes à distance (ouverture de vannes, démarrage de pompes, demande de transfert d'une soute à une autre, ...). La mise en oeuvre de boucles de vérifications et de validations d'un tel système demande de pouvoir mettre en oeuvre, d'une part, des tests utilisateurs pour s'assurer de l'adéquation avec les besoins de l'utilisateur final ; et d'autre part, de garantir le bon fonctionnement de la chaîne de contrôle-commande (c'est-à-dire tester le comportement dynamique) pour s'assurer de l'adéquation du système comme un tout.

Ces tests nécessitent de disposer d'un prototype ou d'une maquette du système physique à superviser. Or la mise à disposition de ces dispositifs de tests reste couteuse, et requiert un maintien en conditions opérationnelles. De plus, lors de ces tests, l'existence de défauts de conception peut, le cas échéant, endommager le matériel (et par conséquent engendrer des coûts importants) ou bien mettre en danger les opérateurs.

Il apparaît donc une contradiction, pour réduire les coûts et les délais de conception, il est important de pouvoir détecter au plus tôt les erreurs de conception afin de les corriger à

moindre coût, or la mise en oeuvre vérification et validation engendre des coûts et des délais de conception potentiellement importants.

A ce titre, nos travaux visent à faciliter la mise en oeuvre de vérifications et de validations dès le début de la conception de systèmes de conduite, dans un contexte industriel. Un intérêt est porté sur l'utilisation de techniques de simulation afin d'utiliser un prototype virtuel pour effectuer les tests. Ceci devrait permettre de pouvoir effectuer des tests tout au long de la conception, sans dangers pour le matériel ou les opérateurs, tout en permettant d'examiner les réactions face à certains aléas (comme par exemple des pannes).

Le manuscrit est structuré en trois parties. Une **première partie** est consacrée au *contexte et problématique* des travaux. Celle-ci est constituée de trois chapitres, un premier présentant le contexte tant scientifique qu'industriel des travaux, un second concernant les apports de la simulation. Enfin, un dernier chapitre conclut cette première partie, en présentant la problématique des travaux ainsi que nos orientations, dévoilant ainsi de façon plus détaillée l'organisation des deux autres parties. La **seconde partie** concerne *nos propositions théoriques* en réponse aux trois verrous identifiés. Chacun de ces verrous traitant des problèmes spécifiques, nous avons choisi de consacrer à chacun d'entre eux un chapitre qui, outre la contribution concernée, contient un état de l'art spécifique. La **troisième partie** du manuscrit concerne quant à elle l'*application de nos propositions théoriques.*

Partie A -

Contexte et Problématique

Dans cette première partie du manuscrit, nous nous proposons, au travers de trois chapitres, de présenter le contexte et les enjeux de ces travaux de thèse pour conclure par la problématique résultante.

Le Chapitre 1 a pour objectif de présenter le contexte scientifique et industriel, afin d'en faire émerger les enjeux. Dans un premier temps, nous présentons le contexte général de ces travaux à savoir la conception de systèmes, en général puis concernant un type particulier de système : les systèmes de conduite de procédés. Dans un second temps, ces travaux faisant l'objet d'une thèse CIFRE, nous présentons le contexte industriel concernant la conception de systèmes de conduite de navire. Puis nous concluons ce chapitre sur un enjeu majeur dans un contexte industriel : le besoin de vérifications et validations au plus tôt.

Le Chapitre 2 s'intéresse à l'apport de la simulation en réponse au besoin de vérifications et validations. Dans un premier temps, nous présentons la simulation en tant que support pour l'ingénierie, c'est à dire comme un outil permettant d'effectuer des tests. Puis, nous nous intéressons à la simulation en tant qu'activité, c'est-à-dire que l'on s'intéresse au processus qui permet de construire et développer une simulation. Enfin, un bilan sur l'apport de la simulation termine ce chapitre.

Le Chapitre 3 conclue cette première partie du manuscrit, une synthèse des précédents chapitres permettra d'exposer les objectifs de nos travaux, les défis à relever et le schéma directeur des travaux.

1

Contexte scientifique et industriel

Sommaire

1.1 Conception de systèmes

> Comme le rappelle Le Moigne, dès 1977 Edgar Morin écrivait : « *Le système a pris la place de l'objet simple et substanciel, et il est rebelle à la réduction en ses éléments. L'enchaînement de systèmes de systèmes brise l'idée d'objet clos et auto-suffisant. On a toujours traité les systèmes comme des objets ;* ***il s'agit désormais de concevoir les objets comme des systèmes.*** *Dès lors il faut concevoir ce qu'est un système.* » [Le Moigne, 2015].

Avant de s'intéresser aux méthodes et outils de conception des systèmes (comme le laisse présager le titre), l'objet de la première section concernera la notion de *système*, ce qui nous amènera à présenter les problématiques liées à la conception de systèmes dans un deuxième temps. Puis, nous nous intéresserons à un type particulier de systèmes : les systèmes de

conduite de procédé. Pour finir, nous considèrerons un cas industriel, à savoir la conception de système de conduite de navire.

1.1.1 La notion de *système*

Tâchons, en premier lieu de répondre à la question : Qu'est-ce qu'un *système* ? Pour y répondre, appuyons-nous sur la définition ci-dessous.

Définition 1.1 (Système [Fiorèse and Meinadier, 2012])
Par définition, tout système est constitué d'un ensemble d'éléments dont la synergie est organisée pour répondre à une finalité dans un environnement donné.

Un *système* **répond** donc à **une finalité** (sa raison d'être), il fournit des services à son environnement. L'interaction entre le système et l'environnement se fait par des échanges de flux, que ce soit de matière, d'énergie ou d'information au travers d'interfaces. Le *système* agit sur ces flux : il va transformer, consommer et/ou utiliser les flux entrants, et va générer des flux sortants. Les éléments qui constituent le système peuvent être de nature très différente et être eux-mêmes des systèmes (cf Figure 1.1).

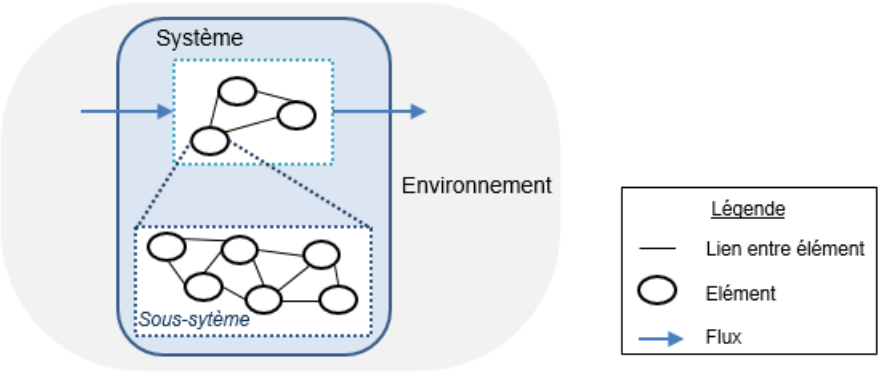


FIGURE 1.1 – La notion de système

La notion de *système* permet ainsi d'appréhender la complexité du réel [Morin, 2015], de part le nombre potentiellement élevé de constituants, de leur nature potentiellement très hétérogène et des fortes interactions entre les constituants et l'environnement. Au travers de cette notion de *système*, nous ne cherchons pas à découper un "problème" (qu'il soit de conception ou non) en plusieurs petits "problèmes" élémentaires dont la résolution (indépendamment des autres) de chacun d'entre eux permettrait d'obtenir une solution (par une addition des solutions élémentaires trouvées). Non, nous cherchons à rendre intelligible le complexe, cette imbrication d'interactions qu'il est difficile d'appréhender comme un tout. L'approche système permet de mettre en évidence ces interactions, et par un jeu de "zoom" de pouvoir regarder ces interactions à différents niveaux de détails (de granularité). En zoomant, on passe d'une représentation de type boîte noire, à une représentation de type boite blanche, faisant ainsi apparaître des interactions internes.

Puisqu'un système a une raison d'être, la conception/construction d'un système répond à un besoin (ou plus vraisemblablement, à des besoins). Ces systèmes, que l'humain va chercher

à concevoir et construire, peuvent être de nature diverse (tant matérielle qu'organisationnelle). Avec l'évolution des techniques et des technologies, l'humain cherche à concevoir des systèmes de plus en plus complexes. On parle d'ailleurs de conception de systèmes complexes et d'ingénierie des systèmes complexes. Un exemple de définition de ces nouveaux systèmes à concevoir est donné ci-dessous.

Définition 1.2 (Système *complexe* [Faisandier, 2011])
Un système est caractérisé par un ensemble de **constituants** (matériels technologiques, logiciels, opérateurs humains, matériaux, procédures, services) ; les constituants sont en forte interaction, et écoulent des **flux** de matière, d'énergie et d'information dans un environnement ou contexte donné. Cet ensemble satisfait des **besoins**, des attentes ; il accomplit une **mission** assortie d'**objectifs** prescrits permettant d'atteindre ou de répondre à une **finalité**.

L'Association Française d'Ingénierie Systèmes (AFIS) définit un système (complexe) comme "*un ensemble d'éléments qui interagissent entre eux et avec leur environnement en fonction d'un but (finalité,mission)*". La structure d'un système est donnée par ses éléments et leurs liens. Son comportement est déterminé par la mise en relation dynamique de ses éléments. Enfin, de par la cohésion d'un système, l'ensemble des effets qu'il produit est supérieur à la somme des effets de ses parties prises indépendamment [Daniel Allegro et al., 2014].

Ainsi, un *système* interagit avec l'environnement dans lequel il est placé et évolue pour s'adapter aux changements de celui-ci dans le but d'assurer sa finalité. Pour tenir compte des évolutions de l'environnement, le *système* dispose de constituants qui **pilotent son fonctionnement**. Cette partie de pilotage (Figure 1.2) lui permet de pouvoir s'adapter à son environnement et ainsi de poursuivre sa "mission", qu'il soit piloté par une intervention humaine ou de façon automatisée.

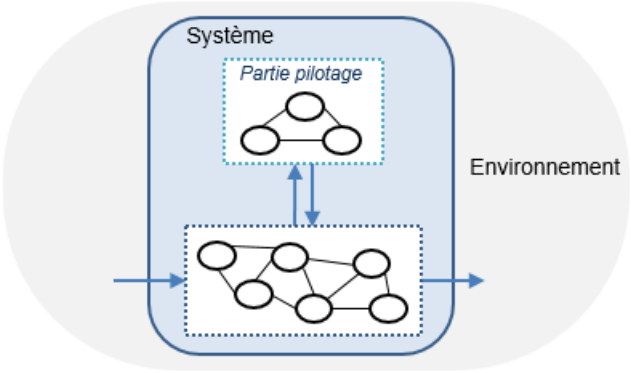


FIGURE 1.2 – Un système

Par conséquent, lors de la conception d'un système on cherche à garantir sa capacité à poursuivre sa "mission", quelles que soient les évolutions de l'environnement.

1.1.2 La conception d'un *système*

Afin de concevoir, réaliser et mettre à disposition un système, les activités correspondantes doivent être organisées, exécutées et coordonnées. Cela requiert la mise en place d'un système *pour faire*, dont la dominante est plutôt organisationnelle, contrairement au système *à faire*

dont la dominante est plutôt technique [Fiorèse and Meinadier, 2012]. Selon ces auteurs, les définitions sont les suivantes.

> Le *système à faire* est initialement un concept, une abstraction. Le résultat concret de sa réalisation est un "produit" (produit-système) ayant les caractéristiques d'un système et répondant à une "définition" qui en précise toutes les caractéristiques techniques, de production, d'exploitation, de maintenance, de retrait de service, etc.
>
> Pour organiser, exécuter et coordonner toutes les activités qui conduisent de l'énoncé de la finalité à la réalisation et à la mise à disposition du système à faire, il est nécessaire de mettre en place un système doté de ressources humaines et techniques : le *système pour faire*. Ce système est organisé sous forme d'un ou plusieurs projets.

Le système *pour faire* repose de plus sur l'utilisation d'outils numériques, ne serait-ce que pour supporter la conduite de projets. Ainsi, quelle que soit la nature du système *à faire*, le système *pour faire* mis en place peut-être vu comme un système ayant une forte interaction entre humains et technologies. Ce sont, en effet, bien les humains qui sont acteurs de ces activités de conception et réalisation, activités qui sont supportées par l'utilisation de technologies. Il y a donc en plus d'une dimension sociologique, une dimension technique à prendre en compte. Le système *pour faire* est ainsi un système sociotechnique, dont une définition est ci-dessous.

Définition 1.3 (Système sociotechnique [Baxter and Sommerville, 2011])
Un système sociotechnique est caractérisé par une interaction complexe entre les humains, les machines et l'environnement dans lequel il évolue.

Les causes à l'origine de l'introduction de défauts lors des phases de conception identifiées par l'AFIS, illustrent bien ce besoin d'articulation entre dimension technique et sociologique. Parmi les causes identifiées par l'AFIS [Fiorèse and Meinadier, 2012], on peut citer, par exemple :

- des besoins insuffisamment exprimés,
- des attentes de certaines parties prenantes mal perçues ou non formalisées,
- des spécifications imprécises et incomplètes
- des solutions qui ne sont pas justifiées ou non validées,
- une mauvaise définition des responsabilités et rôles avec les acteurs,
- une communication entre acteurs qui n'est pas maîtrisée.

Ces causes mettent en lumière les défis à relever dans le cadre des activités de conception. [Cabrera et al., 2010] ont très justement exprimé ces défis dans le cadre de la conception de systèmes mécatroniques, activité multidisciplinaire par essence. Il s'agit selon eux, d'assurer un travail coopératif et une communication efficace parmi les ingénieurs, de disposer d'outils et de méthodes supportant la conception multidisciplinaire, ainsi qu'un support de la conception du logiciel de commande (partie pilotage du système).

Ainsi, on voit apparaître plusieurs besoins :

- **s'assurer** que les **exigences** et les **spécifications** soient **complètes** (par rapport aux clients et utilisateurs finaux)
- **mettre en place** des **vérifications** et des **validations** tout au long de la conception

- **améliorer la communication** au sein d'équipes pluridiciplinaires, et des parties prenantes.

Pour faire face à ces problématiques, certains changements sont donc apparus, notamment au travers de l'ingénierie système qui fournit un certain nombre de préconisations. L'ingénierie système est, selon l'AFIS, « une démarche méthodologique générale qui englobe l'ensemble des activités adéquates pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client tout en satisfaisant l'ensemble des parties prenantes ». Elle préconise, entre autres, des cycles de conception courts et itératifs. De façon générale, les activités de base de l'ingénierie système consistent à [Foures, 2015] :

1. Définir la mission du système (phase d'analyse du besoin).
2. Établir les exigences à satisfaire (phase de spécification du système).
3. Élaborer une conception.
4. Itérer la démarche pour les sous-systèmes.
5. Vérifier que les exigences soient respectées.
6. Valider que la mission effectuée satisfasse le client.

Traditionnellement, le système *pour faire* était basé sur une coordination séquentielle des activités de conception, réalisation et déploiement. Les activités de conception suivaient, elles aussi, un processus séquentiel : on attendait d'avoir fini une étape pour passer à la suivante. On attendait, en particulier, d'avoir fini la conception d'une sous-partie pour concevoir la suivante. Ainsi, une fois que toute la conception des sous-parties du système *à piloter* était finie, on commençait la conception de la partie qui *pilote* le système. Le cycle de conception était donc relativement long, qui plus est, lorsqu'en fin de cycle des erreurs de conception étaient découvertes.

Il existe actuellement différents types de cycle de conception et de développement d'un système. Le plus répandu étant le *cycle en V* [Mooz and Forsberg, 2006], qui fait notamment apparaître les étapes de vérification et validation [INCOSE, 2011]. Une approche descendante (partie gauche du V) permet à partir de l'analyse des besoins, d'arriver aux étapes de développement et de réalisation des composants du système (bas du V). Puis au travers d'une approche ascendante (partie droite du V), ces composants sont intégrés (phase d'assemblage) et testés, pour ensuite arriver à la dernière étape (appelée qualification opérationnelle ou recette). Dans cette dernière étape il s'agit de vérifier que le système est conforme aux besoins opérationnels [Foures, 2015]. Bien que les étapes du cycle en V soient représentées de façon séquentielle, on n'attend plus qu'une étape soit finie pour démarrer la suivante. En effet, aujourd'hui, afin de réduire les coûts et les délais de conception, on vise une approche de conception, où les activités démarrent le plus tôt possible et s'effectuent simultanément.

Issu du génie locigiciel, on peut aussi citer le *cycle en spirale* [Boehm, 1988]. Ce processus de conception reprend les étapes du cycle en V, mais avec plusieurs itérations du cycle complet (autant de fois que nécessaire). Chaque itération du cycle est composée de 4 grandes étapes : *1)* Déterminer les objectifs, choix et contraintes ; *2)* Evaluer les choix, identifier et résoudre les risques ; *3)* Développer, tester et valider ; *4)* Planifier les phases suivantes. De plus, le cycle

en spirale introduit l'utilisation de prototypes, notamment pour pouvoir expérimenter avec les utilisateurs.

L'intérêt de ces types d'approches est d'avoir des **cycles de conception courts et itératifs**, intégrant des activités de vérification et de validation. En effet, l'Ingénierie Système prône une mise en place d'activités de vérification et de validation tout au long du cycle de conception.

Toutefois, on retrouve différentes définitions des notions de *vérification* et de *validation* dans la littérature. Une synthèse des différentes définitions a notamment été proposée dans [Maropoulos and Ceglarek, 2010]. Pour ces auteurs, la *vérification* est « le processus de contrôle de qualité qui est utilisé pour évaluer si oui ou non un produit, un service ou un système respecte les règles, les spécifications ou bien les conditions imposées au début des phases de conception ou de développement ». Et la *validation* est « le processus d'assurance de qualité établissant les preuves qui permettent de fournir un degré d'assurance qu'un produit, un service ou un système respecte bien les exigences d'utilisation prévues ». Des définitions semblables sont aussi données dans [Meinadier, 2009], à savoir : la *vérification* est « la confirmation par examen et apport de preuves objectives que les exigences spécifiées sont satisfaites ». La *validation*, quant à elle, est « la confirmation par examen et apport de preuves objectives que le système répond aux besoins de son acquéreur dans l'environnement opérationnel et les conditions d'emplois qu'il a définis ».

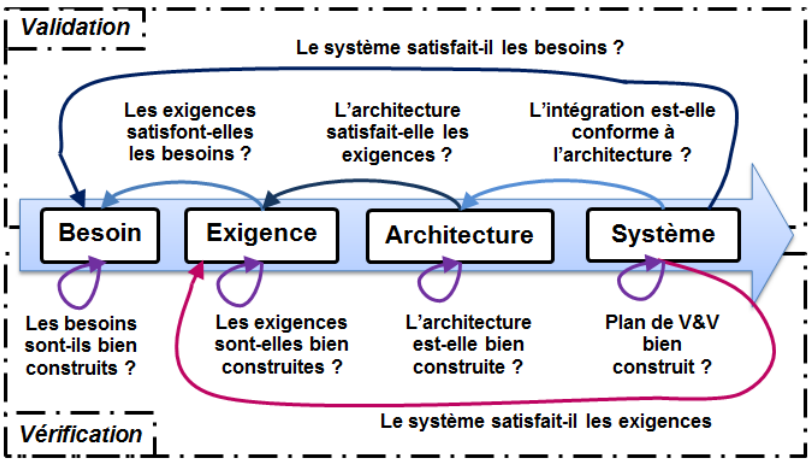


FIGURE 1.3 – Activité de V&V pendant le cycle de conception [Foures, 2015]

Ainsi, la **vérification** consiste à déterminer si le système est conçu correctement (par rapport aux spécifications). Et la **validation** consiste à déterminer si c'est le bon système qui a été conçu (par rapport au besoin). La Figure 1.3, reprise des travaux de [Foures, 2015], montre le processus de vérification et validation pendant le cycle de conception.

L'ingénierie Système, outre la préconisation de cycles de conception courts et itératifs, s'intéresse aussi aux méthodologies pour supporter les activités de conception. Ceci sera l'objet de la section suivante.

1.1.3 Une méthodologie de conception basée sur les modèles

Les activités de conception s'appuient sur des représentations du système à concevoir, de l'environnement dans lequel il sera placé et de solutions : elles reposent sur des *modèles*. En effet, les concepteurs raisonnent sur des *modèles*. Ces modèles sont des représentations de la réalité, à travers différents points de vue et sont construits dans une intention particulière. Un *modèle* n'est donc pas une représentation de la réalité "objective", puisque cela dépend du regard porté sur la réalité d'une part, et d'autre part, de l'interprétation des observations de la réalité qui sont faites par l'observateur. Il n'y a donc pas de "vrai" modèle, ni de modèle unique de la réalité. Par contre un modèle, peut être correct et complet par rapport à un objectif d'utilisation.

Les modèles peuvent ainsi prendre diverses formes de représentations (cela peut être des schémas, comme des systèmes d'équations). Ces représentations, qu'elles soient graphiques ou non, sont construites à l'aide de "symboles", qui, mis ensemble, prennent une signification. Ainsi lorsque l'on construit un modèle on s'appuie sur une codification de l'information : un ensemble de symboles et de règles d'interprétation de ces symboles selon leur organisation. Un modèle est donc exprimé au travers d'un *langage*, qu'il soit pré-existant (langage mathématique, schéma normalisé) ou bien défini pour l'occasion. Cette notion de *langage de modélisation* est d'autant plus importante, que le modèle ne sert pas uniquement à celui qui le construit, il sert aussi à communiquer avec d'autres personnes.

Ainsi, un modèle est conçu pour résoudre un problème. Un système possède donc une infinité de modèles qui sont adaptés pour différentes tâches. La notion de tâche est capturée par le niveau d'abstraction, qui est déterminé par la perspective que l'on a du système, le problème à résoudre et l'expérience du modélisateur [Mosterman and Vangheluwe, 2004]. Toute modélisation se fait donc selon un objectif.

D'après [Hardebolle and Boulanger, 2009], l'objectif de modélisation peut se représenter selon le quadruplet : $< domaine, abstraction, vue, activité >$

- où le *domaine* correspond au domaine technique ou métier du système/sous-système considéré,
- l'*abstraction* correspond au niveau de détail pour le système/sous-système étudié,
- la *vue* correspond à l'aspect sous lequel on considère le système/sous-système,
- et l'*activité* correspond à l'activité du cycle de conception où l'on se situe.

Selon l'objectif de la modélisation, différents paradigmes peuvent ainsi être utilisés pour la modélisation. Un paradigme de modélisation est un état d'esprit pour la modélisation. En d'autres termes, cela définit le ou les concepts à travers lesquels on voit le "système" à modéliser. Un paradigme de modélisation peut donc être implémenté par différents langages ou formalismes.

Ainsi, la conception d'un système passe par des activités de modélisation, et l'utilisation de modèles. Les activités de conception reposant sur les modèles, l'Ingénierie Système promeut l'utilisation de méthodologies de conception basées sur les modèles.

1.1.3.1 Ingénierie Système Basée sur les Modèles

L'Ingénierie Système Basée sur les Modèles, *Model-Based System Engineering* (MBSE), est une méthodologie visant à soutenir les activités pendant tout le cycle de vie d'un système, via l'utilisation de modèles. Ainsi une méthodologie MBSE fait référence à une méthodologie utilisée pour soutenir l'Ingénierie Système dans un contexte "basé sur les modèles" ou "dirigé par les modèles", en opposition aux méthodologies ("traditionnelles") qui étaient basées sur l'utilisation de documents.

Mais qu'est-ce qu'une *méthodologie ?* Pour répondre, nous nous appuierons sur les définitions de [Rochet, 2007] ci-dessous.

Une **méthodologie** est définie comme une collection de processus, de méthodes et d'outils reliés. Une méthodologie est essentiellement une "recette" et peut être comprise comme l'application de processus, méthodes et outils relatifs à une classe de problèmes qui ont quelque chose en commun.

Un **processus** est une séquence logique de tâches réalisées pour atteindre un objectif particulier. Il définit ce qui est à faire sans préciser le "comment faire". La structure d'un processus peut se présenter sous différents niveaux d'agrégation pour permettre les analyses et répondre aux différents besoins d'aide à la décision.

Une **méthode** est attachée aux techniques de réalisation d'une tâche. Elle définit le "comment faire" de chaque tâche (dans ce contexte, les mots "méthode", "technique", "pratique" et "procédure" sont souvent interchangeables). A tous les niveaux, les tâches de processus sont réalisées en utilisant des méthodes. Cependant, chaque méthode est aussi, elle-même, un processus avec sa séquence de tâches à réaliser selon des méthodes particulières. En d'autres termes, le "comment" d'un niveau d'abstraction devient le "quoi" du niveau inférieur.

Un **outil** est attaché à une méthode particulière, pour augmenter l'efficacité de réalisation de la tâche. Le rôle d'un outil est de faciliter le "comment".

D'autre part, une méthodologie est mise en oeuvre dans un environnement [1], l'environnement du projet. Cet environnement a pour but d'intégrer et de supporter l'usage des outils et des méthodes utilisés sur ce projet [Estefan et al., 2007]. En d'autre termes, un environnement active (ou désactive) ce qui est à faire et le "comment faire".

Il ressort dans la revue de la littérature [Estefan et al., 2007] que les méthodologies MBSE reposent majoritairement sur l'utilisation d'un langage de modélisation tel que UML (*Unified Modeling Language*) [2] et/ou SysML. Afin de pouvoir représenter un système selon ses différentes facettes, et de faciliter la communication au sein des équipes hétérogènes, l'INCOSE a défini un langage de modélisation en se basant sur UML : SysML (*Systems Modeling Language*). Ce langage permet notamment de représenter les exigences, la structure,

1. Un environnement est composé du milieu, des objets externes, de conditions (sociales, culturelles, personnelles, physiques, organisationnelles ou fonctionnelles), ou de facteurs qui influencent les actions d'un objet, d'un individu ou d'un groupe de personnes [Estefan et al., 2007].
2. UML est un langage de modélisation unifié issue du génie logiciel.

le comportement, les propriétés et les interconnexions d'un système au travers de différents diagrammes.

Toutefois, la mise en oeuvre d'une méthodologie basée sur les modèles, au sein des activités de conception, requiert des transformations de modèles. A ce titre, l'Ingénierie Dirigée par les Modèles (IDM), ou *Model Driven Engineering* en anglais, se focalise sur celles-ci. Ce sera ainsi l'objet de la section suivante.

1.1.3.2 Ingénierie Dirigée par les Modèles (IDM)

L'IDM, issue du génie logiciel, est apparue pour répondre à des problématiques liées à la conception et au développement de systèmes complexes. En effet, la conception et le développement de tels systèmes font intervenir différents corps de métiers très variés, et doit prendre en compte des aspects de sûreté de fonctionnement. L'objectif de l'IDM est de fournir autant de modèles que nécessaire pour exprimer chacune des préoccupations des utilisateurs/concepteurs/architectes ..., de manière séparée.

La notion clé est celle de *modèle*. Contrairement aux approches orientées objets où *tout est objet*, ici *tout est modèle* (voir Figure 1.4). Un modèle est une représentation d'un système (relation *representationDe*). De plus, un modèle est défini par un langage de modélisation (c'est ce qui est appelé un *métamodèle*). Tous les éléments du modèle sont définis par le métamodèle. De ce fait, un modèle doit être conforme à son métamodèle (relation *conformeA*).

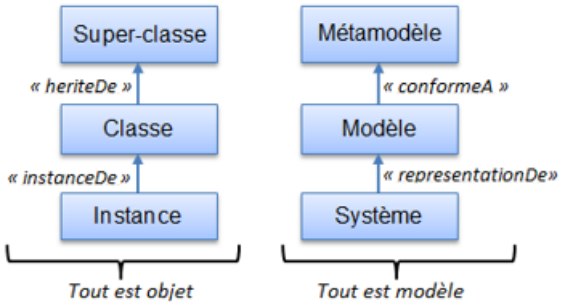


FIGURE 1.4 – Parallèle entre approche orientée objet et IDM

Définition 1.4 (Métamodèle [Combemale, 2008])
Un métamodèle est un modèle qui définit le langage d'expression d'un modèle, c'est à dire qu'il définit le langage de modélisation.

Puisque l'on considère que *tout est modèle*, le code d'un programme est vu comme un modèle exécutable. Le principe est d'utiliser la transformation de modèles afin d'obtenir un modèle exécutable (code exécutable). Un autre principe clé de l'IDM est d'utiliser autant de langages de modélisation que nécessaire. Ces langages de modélisation sont spécifiques au domaine, d'où la dénomination anglaise de *Domain Specific Language* (DSL).

Définition 1.5 (Méta-modélisation)
La *méta-modélisation* est l'activité consistant à définir le méta-modèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser.

La méta-modélisation est une activité qui consiste à modéliser (ou définir) les formalismes de modélisation. Par conséquent, elle permet de définir des classes de modèles, puisque cela définit des langages de modélisation [Mosterman and Vangheluwe, 2004].

Il existe plusieurs catégories de transformations. Celles *endogènes* ou *exogènes* qui concernent le langage de modélisation, et celles *horizontales* ou *verticales* qui concernent le type de modèle, spécifique ou non à une plate-forme d'exécution. Lors d'une transformation *endogène*, le modèle source (d'entrée) et celui cible (de sortie) sont conformes au même méta-modèle. Lors d'une transformation *exogène* les modèles source et cible sont conformes à leur propre méta-modèle. Une transformation *horizontale* (cf Figure 1.5) correspond à une transformation d'un modèle PIM (*Platform Independent Model*) vers un autre modèle PIM ou d'un modèle PSM (*Platform Specific Model*) vers un autre modèle PSM. Tandis qu'une transformation *verticale* correspond à une transformation d'un modèle pérenne PIM vers une modèle PSM, ou l'inverse.

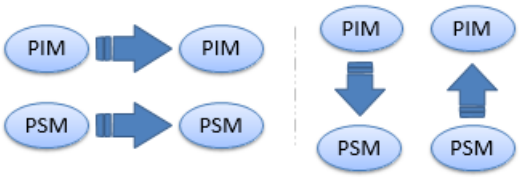


FIGURE 1.5 – Transformation Horizontale (à gauche) et transformation Verticale (à droite)

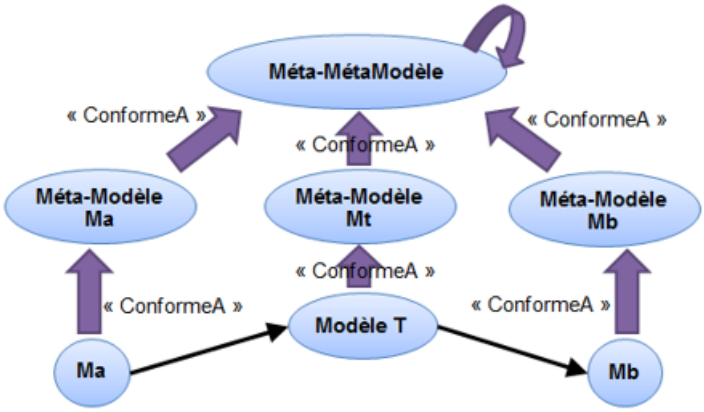


FIGURE 1.6 – Principe de la transformation de modèle

La transformation d'un modèle en un autre se fait en utilisant un modèle de transformation. De façon générique, la Figure 1.6 illustre le principe de transformation de modèle du modèle *Ma* en un modèle *Mb* via l'utilisation d'un modèle de transformation *T*. Les transformations *horizontales endogènes* permettent la restructuration (amélioration des performances en conservant la même sémantique), la normalisation ou la simplification (réduction de la complexité syntaxique du modèle) et l'intégration de patrons. Les transformations *horizontales exogènes* permettent la migration de logiciels et la fusion de modèles. Les transformations *verticales endogènes* permettent le raffinement (enrichissement ou modification de l'information dans un modèle pour en améliorer certains aspects qualitatifs). Enfin, les transformations *verticales exogènes* permettent la génération (transformation d'un niveau d'abstraction vers un niveau inférieur) et la rétro-conception (transformation d'un niveau d'abstraction vers un

plus élevé).

L'Ingénierie Système Basée sur les Modèles et l'Ingénierie Dirigée par les Modèles sont liées. Toutes les deux partagent la notion de *modèle* comme un élément central. Cependant, elles ne se focalisent pas sur les mêmes choses. En effet, dans un cas on s'intéresse aux méthodologies de conception, dans l'autre on se focalise sur les échanges (transformations) de modèles, c'est à dire comment passer d'un modèle à l'autre. IDM et MBSE n'interviennent donc pas au même niveau, mais ne peuvent exister l'une sans l'autre.

1.1.4 Bilan

Aujourd'hui, avec l'évolution des techniques et des technologies, on cherche à concevoir des artefacts de plus en plus complexes. Ainsi on ne cherche plus à concevoir des objets mais des systèmes. La notion de *système* permet de mieux appréhender la complexité, en mettant en évidence les interactions internes au système mais aussi externes (celles avec l'environnement). Cette notion de système permet aussi d'appréhender l'organisation, l'exécution et la coordination des activités de conception (au sens large) via la mise en place d'un système pour faire.

L'évolution de la complexité des systèmes à concevoir nécessite d'adapter les démarches de conception. En effet, l'approche de conception traditionnelle de type séquentielle ne permet pas d'appréhender le système comme un « tout ». Les choix de conception lors d'une étape peuvent avoir de grandes répercutions sur l'ensemble du système, et ne peuvent être détectés qu'en fin de conception (dans les phases d'intégration). Pour appréhender le système comme un "tout" une approche de **conception simultanée** est donc nécessaire.

De plus, avoir des **cycles courts et itératifs** permet d'intégrer des boucles de vérifications/validations, notamment pour s'assurer que le problème est bien posé et que la solution répond au problème (et aux besoins). Ainsi, les rectifications peuvent être faites plus tôt, ce qui diminue les coûts et les délais de re-conception.

D'autre part, l'activité de conception s'appuie sur l'utilisation de *modèles*. Cette notion de *modèle* est importante puisqu'elle aide les concepteurs à raisonner et à communiquer entre eux. On passe ainsi d'une ingénierie basée sur les documents à une ingénierie à base de modèles. On ne communique plus au travers de documents mais via des modèles.

Pour faciliter les échanges entre concepteurs et ainsi limiter les erreurs d'interprétation, l'idée est de ré-utiliser des représentations qui ont été construites dans le cadre des activités de conception. Toutefois, chaque discipline a ses propres représentations (qui ne sont pas forcément intelligibles par tout le monde). L'utilisation de techniques de l'IDM permet d'automatiser le passage d'un modèle à un autre (transformation de modèle), à moindre coût et en limitant l'introduction d'erreurs.

Au travers de cette section, nous nous sommes intéressés aux principes généraux de la conception de systèmes afin de définir le contexte général de nos travaux. De façon générale, les cycles de conception et de développement doivent être courts et itératifs, dans lesquels il faut s'assurer que les exigences et les spécifications du système soient complètes (par rapport

au client et à l'utilisateur final), et mettre en place des processus de vérification et de validation tout au long du cycle de conception. D'autre part, l'évolution de la complexité du système *à faire* requiert des équipes pluridisciplinaires, au sein desquelles un réel besoin d'amélioration de la communication et de l'interopérabilité est présent.

A ces problématiques générales de conception, s'ajoutent des problématiques propres à la nature des systèmes à concevoir. Ainsi dans la prochaine section, on s'intéressera à la conception d'un type particulier de système : la conduite de procédé.

1.2 Conception de systèmes de conduite de procédés

L'évolution de la complexité des systèmes peut être perçue par le niveau d'automatisation des systèmes que l'on cherche à concevoir. Comme évoqué précédemment, un *système* pour s'adapter à son environnement doit évoluer, et dispose pour cela de constituants qui pilotent son fonctionnement (partie pilotage du système). Cette partie de pilotage peut être vue comme un élément du système qui est intégré à la partie qu'elle pilote, comme dans le cas des systèmes mécatroniques. Elle peut aussi être vue comme un sous-système, que l'on pourrait qualifier de système *de pilotage*, et qui peut être plus ou moins automatisé. Considérons le système représenté sur la Figure 1.7, le système de conduite correspond au système de pilotage du système. C'est ce qui permet de piloter le procédé et d'adapter le comportement du procédé en fonction des évolutions de l'environnement (que ces évolutions soient prévues ou non).

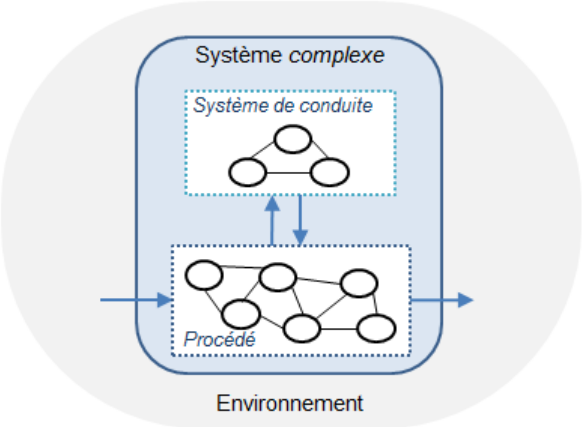


FIGURE 1.7 – La place d'un système de conduite dans le système

Les années 1980 ont connu une automatisation massive des systèmes, cependant cette automatisation de la commande, prévue pour un fonctionnement normal, a montré ses limites et défauts [Combacau, 1998]. En effet, dans la réalité, des aléas peuvent survenir lorsque le système est en exploitation. Du fait de ces aléas, le système peut ne plus se trouver dans les conditions normales pour lesquelles la commande a été prévue, ce qui peut conduire à des évolutions catastrophiques du système automatisé. L'automatisation des systèmes ne consiste donc pas uniquement à automatiser la commande d'un procédé, mais nécessite également de surveiller le système et de prendre des décisions.

Ainsi, avec l'automatisation des systèmes, le procédé est sous un contrôle partiel de l'opérateur (humain) : l'opérateur supervise la conduite du procédé. Le système *de pilotage* peut donc être vu comme un système Homme-Machine. L'Homme et la Machine y sont considérés comme les deux composants du système et sont engagés conjointement dans la réalisation d'une tâche, cette dernière étant celle du système dans son ensemble [Folcher and Rabardel, 2004].

Traditionnellement, le système *à faire* était considéré uniquement dans sa dimension technique. L'humain n'était alors qu'un "simple" utilisateur, perçu comme une entité réalisant une tâche prescrite. Au contraire, aujourd'hui l'humain est considéré comme un système qui doit coopérer avec la Machine afin d'atteindre un but (celui du système Homme-Machine). En effet, la raison d'être d'un opérateur humain ne peut pas être définie par son rôle au sein d'un système. L'opérateur humain, bien qu'il soit engagé conjointement avec la machine dans la réalisation d'une tâche, poursuit des buts qui lui sont propres. L'interaction entre l'homme et la machine, compte tenu d'un environnement donné, n'est donc plus uniquement considérée au travers d'une dimension technique : une dimension humaine doit aussi être prise en compte lors de la conception.

Afin de faire émerger les problématiques et enjeux liés à la conception de ces systèmes, nous reviendrons tout d'abord sur ce qu'est un *système de conduite*. Ensuite, nous nous intéresserons au caractère adaptable du système, caractéristique essentielle pour pouvoir évoluer dans un environnement dynamique, au travers des aspects de sûreté de fonctionnement, puis de la prise en compte du facteur humain. Et enfin, nous nous intéresserons aux techniques de tests permettant de vérifier le comportement du système conçu.

1.2.1 Le système de conduite de procédé

Le but d'un système de conduite, illustré par la Figure 1.8, est de superviser le bon fonctionnement d'un procédé. Pour cela, il faut, d'une part, surveiller l'état du procédé, et d'autre part, pouvoir agir sur le procédé en envoyant des commandes au procédé. Typiquement, cela correspond à un système de contrôle-commande de type SCADA (*Supervisory Control And Data Acquisistion*). Il est composé d'une partie de supervision et d'une partie de commande et de surveillance, au sein desquelles on peut retrouver différents niveaux d'automatisation. Ce type de système a donc une dominante logicielle, contrairement au procédé qui est à dominante matérielle (mécanique).

Avant de s'intéresser plus en détail au système de conduite, il est nécessaire de définir les notions de commande, surveillance et supervision. Nous nous appuierons pour cela sur les définitions issues de [Combacau et al., 2000] et de [Niel and Craye, 2002].

Le rôle de la commande est d'exécuter un ensemble d'opérations (qui, suivant le niveau d'abstraction où l'on se place, peuvent être élémentaires ou non) sur le procédé, en fixant des consignes de fonctionnement en réponse à des ordres d'exécution. Il peut s'agir de réaliser :

- une séquence d'opérations pour réaliser un produit ou un service ;
- une séquence d'actions corrective destinée à restaurer tout ou une partie des fonctionnalités requises pour que le système assure sa mission. La perte d'une partie des

fonctionnalités qui étaient initialement disponibles fait suite à l'occurrence d'une défaillance ;
- des actions prioritaires, et souvent prédéfinies, sur le procédé afin d'assurer la sécurité de l'installation et du personnel ;
- des opérations de test, réglage ou de nettoyage exécutées afin de maintenir le procédé dans un état opérationnel (pour garantir que le système pourra continuer à assurer sa mission).

Le rôle de la surveillance est de recueillir en permanence tous les signaux en provenance du procédé et de la commande, de reconstituer l'état réel du système commandé, de faire toutes les inférences nécessaires pour produire les données utilisées. Que ce soit pour dresser des historiques de fonctionnement, ou le cas échéant, pour mettre en oeuvre un processus de traitement de défaillance. La surveillance est donc limitée aux fonctions de collecte et de traitement de données qui n'ont pas d'action directe sur la commande ou sur le procédé. Elle a un rôle passif par rapport au système de commande et du procédé.

Le rôle de la supervision concerne les prises de décision liées au pilotage du système. La supervision contrôle l'exécution d'une opération ou d'un travail effectué par d'autres, et prend des décisions pour permettre au système d'assurer sa mission malgré les évolutions de l'environnement :
- Pendant un fonctionnement normal, la supervision prend des décisions pour lever l'indécision dans le système de commande (ordonnancement temps-réel, optimisation, mise à jour de la commande et remplacement d'une loi de commande par une autre).
- Quand une défaillance du procédé se produit, la supervision prend toute les décisions nécessaires pour permettre au système de reprendre son fonctionnement normal (réordonnancement, actions de recouvrement, procédures d'urgence,...).

Le concept de supervision s'applique dans un cadre hiérarchisé comportant au moins deux niveaux. A un niveau très local, la supervision peut disparaître dans le sens où la surveillance est intégrée à la commande. De ce fait, tout est prévu et figé à l'avance. A des niveaux très abstraits, la supervision est en revanche prépondérante vis-à-vis de la commande et de la surveillance.

Sur la Figure 1.8, l'interface Homme-Machine (IHM) de supervision, permet de remonter les informations concernant l'état du procédé afin de surveiller le fonctionnement, et de pouvoir agir sur le procédé en envoyant des requêtes (des demandes d'exécution). L'humain n'agit pas directement sur le procédé au travers de l'IHM, il existe une entité intermédiaire entre eux : la partie de contrôle-commande. C'est la partie de contrôle-commande qui interagit directement avec le procédé, elle envoie les commandes au procédé et récupère les informations provenant de l'instrumentation et des capteurs. On voit donc apparaître une chaîne de contrôle-commande, supervisée par l'humain quel que soit le degré d'automatisation.

La structure d'un système de conduite est ainsi hiérarchisée, l'humain étant au plus haut niveau hiérarchique. On voit donc apparaître, dans le système de conduite de procédé, une relation (une dépendance) entre l'opérateur humain et la partie "machine". Bien que l'interaction entre l'homme et la machine (directe) se fasse au travers de l'IHM, ce n'est pas la

seule relation entre l'homme et la machine. On retrouve en effet d'autres niveaux d'interaction, notamment, une coopération entre l'homme et la machine afin d'atteindre les objectifs du système. En allant encore plus loin, du point de vue de l'humain, la machine devient un instrument lui permettant de réaliser son activité (notion d'activité médiatisée par les instruments [Rabardel, 2005]). La relation entre l'humain et la machine au sein du système de conduite est donc multiple (et complexe). Afin que l'opérateur humain puisse superviser la conduite du procédé, il faut, entre autres, que l'on puisse garantir le bon fonctionnement de la partie technique du système.

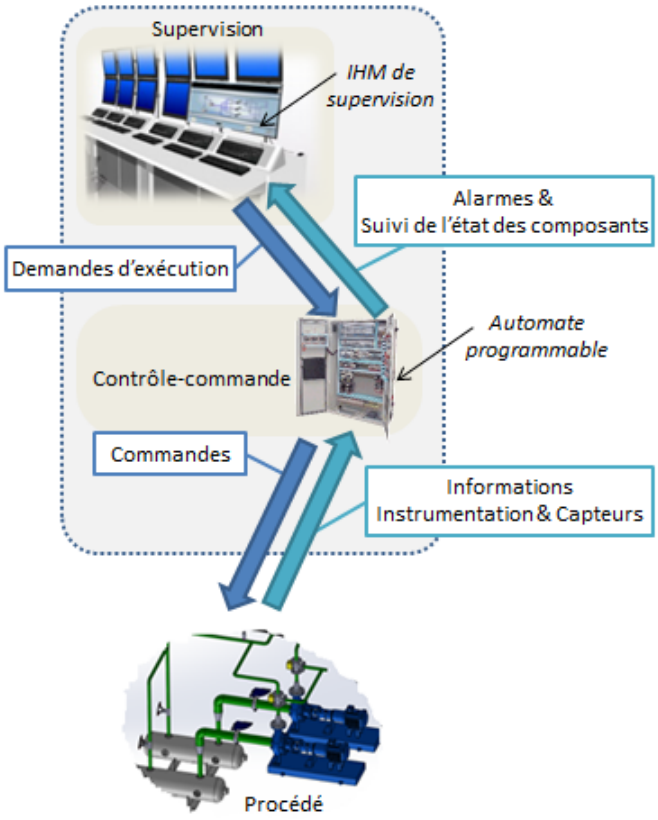


FIGURE 1.8 – Un système de conduite de procédé

Lors de la conception du système, le but est d'éviter l'apparition de défaillances du système pendant l'exploitation. Il faut donc prendre en compte des aspects de sûreté de fonctionnement, afin de garantir un "bon" fonctionnement malgré les évolutions de l'environnement (anticipées ou non).

1.2.2 Sûreté de fonctionnement

La sûreté de fonctionnement (*dependability*) d'un système est son aptitude à délivrer un service de confiance[3] justifiée [Arlat et al., 2006]. Au travers de la notion de confiance, la définition fait ressortir la notion de dépendance entre un système envers un autre comme

3. La confiance est définie comme étant "une dépendance acceptée, explicitement ou implicitement".

étant *l'influence, réelle ou potentielle, de la sûreté de fonctionnement de ce dernier sur la sûreté de fonctionnement du système considéré.*

Trois grandes classes de dysfonctionnement des systèmes automatisés ont été identifiées [Combacau, 1998] : la défaillance d'un composant du procédé commandé, la défaillance du système de commande, et la faute [4] d'un opérateur humain. Afin d'éviter les dysfonctionnements, une nouvelle discipline a vu le jour : la sûreté de fonctionnement. Villemeur désignera cette discipline comme étant "la science des défaillances" [Villemeur, 1988]. On voit ainsi apparaître une notion importante celle de *défaillance*, correspondant au terme anglais *failure*. Une définition de cette dernière est donnée ci-dessous.

Définition 1.6 (Défaillance [Ribot, 2009])
Une **défaillance** correspond à une cessation de l'aptitude d'une entité à accomplir une ou plusieurs fonctions requises.

Dans cette définition, le terme *entité* peut être compris comme "tout élément, composant, sous-système, système, dispositif, équipement, unité fonctionnelle que l'on peut considérer individuellement" [Villemeur, 1988]. Ainsi, après *défaillance* d'une entité, on dit que cette entité est *en panne*. Toutefois, dans la communauté des systèmes continus, *défaillance* et *panne* sont synonymes [Isermann and Balle, 1997]. En effet, une défaillance n'est pas perçue comme un événement (comme un changement d'état) mais comme *l'inaptitude à accomplir une fonction requise*, ce qui correspond à la définition d'une panne donnée par [Villemeur, 1988]. En revanche, dans la communauté informatique [Arlat et al., 2006], la défaillance d'un service correspond à la transition d'un service correct à un service incorrect. La délivrance d'un service incorrect (une fonction requise n'est plus remplie par une entité) est considérée comme une panne de service. De plus, la transition d'un service incorrect à un service correct correspond à une restauration du service.

Ainsi, lors de la conception d'un système on cherche à anticiper ce qui pourrait entraver sa sûreté de fonctionnement. Les causes de défaillance peuvent être diverses, elles peuvent être liées à des erreurs de conception ou de fabrication, comme être liées à une panne d'un élément en exploitation. Différentes terminologies ont ainsi été introduites dans la littérature, au sein des différentes communautés. En particulier, les notions de *défaut*, de *faute* et d'*erreur* peuvent avoir des définitions différentes.

1.2.2.1 Terminologie

Pour Villemeur, un *défaut* est "un écart entre une caractéristique d'une entité et la caractéristique voulue, cet écart dépassant des limites d'acceptabilité" [Villemeur, 1988]. Une définition similaire est donnée par [Niel and Craye, 2002] : "écart existant entre la valeur réelle d'une caractéristique du système et sa valeur nominale". Un *défaut* est donc une déviation d'une valeur réelle d'une caractéristique par rapport à des valeurs limites acceptables. On retrouve aussi le terme de *faute* qui est parfois utilisé dans ce sens là, comme dans [Ribot, 2009]

4. "Action consistant à ne pas prendre en compte ou à mal interpréter une contrainte du cahier des charges ou une contrainte opérationnelle."

où "une faute représente une déviation non acceptable d'au moins une propriété caractéristique ou d'un paramètre du système", ou encore dans [Isermann and Balle, 1997]. Au contraire, pour [Niel and Craye, 2002] une *faute* est "une action, volontaire ou non, dont le résultat est la non prise en compte correcte d'une directive, d'une contrainte exprimée par le cahier des charges". En informatique, une *faute* est considérée comme "la cause adjugée ou supposée d'une erreur", une erreur étant la "partie totale du système qui est succeptible d'entraîner sa défaillance, qui survient lorsque l'erreur affecte le service délivré à l'utilisateur" [Arlat et al., 2006].

Bien que les définitions de *faute* et d'*erreur* diffèrent selon les auteurs, une *erreur* semble toujours être considérée comme la conséquence d'une *faute*. En effet, [Niel and Craye, 2002] considèrent qu'une *erreur* est une "partie du système ne correspondant pas ou correspondant incomplètement au cahier des charges". De plus, puisqu'une faute est une action dont il résulte une non prise en compte d'une contrainte, "en toute logique, une erreur est la conséquence d'une faute". Pour [Isermann and Balle, 1997], une erreur est une déviation entre une valeur mesurée ou calculée (d'une variable de sortie) et la valeur spécifiée ou théoriquement correcte. Dans cette définition, une *erreur* correspond à un *défaut* d'une variable de sortie, tandis qu'une *faute* correspond à un *défaut* d'une propriété caractéristique ou d'un paramètre du système. Il semble donc qu'ici aussi une erreur soit la conséquence d'une faute (ou d'un défaut). Par la suite, nous utiliserons le terme de *défaut* et non de *faute*.

A travers ces définitions, on peut voir apparaître la notion d'erreur *interne*. Une erreur *interne* concerne un ou des composant(s) du système, mais n'impacte pas le service que le système doit délivrer, alors qu'une erreur est associée à une défaillance du système. De plus, on voit apparaître une propagation des défaillances, une chaîne de causes à effets : un *défaut* provoque une erreur *interne*, qui par la suite va elle même générer une erreur entraînant une défaillance ; ou bien du fait de la vulnérabilité du système due à une erreur interne, un *défaut* pourra par la suite générer une erreur qui entraînera une défaillance du système.

1.2.2.2 De la sûreté de fonctionnement à la résilience du système

Pour assurer une continuité de service dans des conditions données, la prise en compte des caractéristiques de sûreté de fonctionnement se traduit par la définition d'une architecture du système qui permet, par exemple, d'effectuer des actions de maintenance tout en garantissant une continuité de service, ou bien de pouvoir continuer à délivrer des services malgré des entités en panne. Afin de développer un système sûr de fonctionnement, différents moyens peuvent être mis en oeuvre [Arlat et al., 2006] : prévention, tolérance, élimination, prévision de défauts (ou fautes au sens de la communauté informatique).

La conception du système de conduite est ainsi une tâche complexe qui nécessite la prise en compte des interactions internes du système, et des évolutions de l'environnement. A ses débuts, il s'agissait principalement de garantir un fonctionnement dans des conditions normales. En effet, au sens strict, "la sûreté de fonctionnement est l'aptitude d'une entité à satisfaire à une ou plusieurs fonctions requises dans des conditions données" [Villemeur, 1988]. Aujourd'hui, il s'agit non seulement de garantir le fonctionnement dans des conditions normales mais

aussi dans des conditions anormales (inattendues). Cependant, il est bien évident que l'on ne peut pas anticiper toutes les évolutions de l'environnement (interne et externe). Comment, dès lors, être en mesure de garantir une sûreté de fonctionnement ?

Le principe est de donner les moyens, la possibilité, au système de pouvoir se modifier (du point de vue de son organisation interne) pour s'adapter au contexte. Pour y parvenir, on introduit des redondances [5] dans le système. Ces redondances peuvent donc se retrouver, tant au niveau logiciel que matériel, et s'appliquer aux différentes entités du système (concernant le système de conduite lui-même ou bien le procédé). Ainsi, un procédé contient des redondances à divers niveaux : éléments, fonctions, sous-système ... Quelle que soit l'architecture retenue pour le système de conduite, vis-à-vis de son rôle au sein du système (piloter le fonctionnement du procédé), il est nécessaire de prendre en compte les redondances du procédé.

En effet, ce sont ces redondances qui permettent au système de conduite de pouvoir modifier et faire évoluer le fonctionnement du procédé, malgré des aléas qui peuvent être internes au système (par exemple, des pannes) ou bien externes (lié à l'environnement du système). La présence de redondances (éléments, fonctions...) permet donc au système de disposer de plusieurs moyens pour atteindre ses objectifs, ce qui lui donne la possibilité de se réorganiser pour continuer à poursuivre ses objectifs. La mise en oeuvre de réorganisation requiert cependant des mécanismes pour pouvoir *détecter le besoin* de réorganisation, pour *décider d'une réorganisation* et pour *effectuer les changements*. Ces mécanismes sont mis en oeuvre dans la partie qui pilote le fonctionnement du système. Ainsi, par exemple, lorsqu'une panne est détectée, le système de conduite agit sur le procédé pour permettre la poursuite de son fonctionnement. Par conséquent, ces mécanismes doivent être prévus lors de la conception et de la réalisation du système.

Néanmoins, puisque l'on ne peut pas anticiper toutes les situations auxquelles sera confronté le système, il faut que le système soit capable de faire face à des situations imprévues, inconnues. On voit ainsi apparaître la notion de *résilience*, qui correspond à la capacité de s'adapter avec succès à des perturbations environnementales inattendues ou à des troubles [Laprie, 2008]. Lors de la conception d'un système de conduite de procédé, on cherche donc à garantir la sûreté de fonctionnement du système (dans son ensemble) ainsi que sa résilience.

Jusqu'à présent nous nous sommes intéressés à la sûreté de fonctionnement à travers une vision technocentrée (défaillance du procédé, défaillance du système de commande). Or, dans le cadre de système de conduite de procédé, l'opérateur humain supervise la conduite. Il n'est donc pas un simple utilisateur d'un système "technique", il est une composante du système (de conduite), caractérisée par une interaction forte avec la partie "technique". Il y a donc une dépendance entre partie technique et partie humaine. L'humain doit donc délivrer un service de confiance justifiée. A ce titre, garantir la sûreté de fonctionnement du système, dans son ensemble, passe par garantir celle du système technique et celle de l'opérateur humain.

De plus, lors de situations inattendues, le rôle de l'opérateur humain est prépondérant : il

5. Une redondance (*redundancy* en anglais) est l'existence, dans une entité, de plus d'un moyen pour accomplir une fonction requise [Villemeur, 1988].

doit intervenir pour permettre au système de faire face. Les propos tenus par [Kolski et al., 1993] en sont une illustration : « *Suite à l'évolution technologique, l'opérateur est de moins en moins impliqué dans les tâches de contrôle manuel. Par contre, il doit de plus en plus réaliser des tâches mentales complexes de résolution de problèmes, et ceci lors de situations dynamiques, aux exigences bien particulières.* »

Ainsi, garantir la sûreté de fonctionnement et la résilience du système de conduite passe également par le fait d'éviter des défaillances humaines. de ce fait, la prise en compte du facteur humain fera l'objet de la section suivante.

1.2.3 Prise en compte du facteur humain

Comme évoqué précédemment, les causes de dysfonctionnements (ou de défaillances) de systèmes automatisés sont liées à des aspects techniques du système et/ou liées à l'intervention humaine. Outre les fautes introduites durant les phases de conception du système, certaines fautes sont introduites en opération, par un opérateur humain. La sûreté de fonctionnement d'un système de conduite de procédé concerne donc également le système dans sa dimension humaine. En effet le système étant supervisé par l'humain, il faut qu'il puisse être en mesure de réaliser sa ou ses tâches. Des aspects de sûreté de fonctionnement par rapport au facteur humain doivent donc aussi être pris en compte.

Les concepts de sûreté de fonctionnement s'appliquant au système technique que nous venons de voir peuvent s'appliquer à l'humain. Toutefois, il est à noter que dans ce contexte, une *défaillance* n'a pas forcément comme conséquence une *panne*. En effet, bien qu'une *défaillance humaine* soit la cessation de l'aptitude d'un opérateur humain à accomplir une mission requise, une *panne humaine* est définie comme une inaptitude *prolongée* d'un opérateur humain à accomplir une mission requise [Villemeur, 1988]. De plus, dans la terminologie associée, une distinction est faite entre la notion d'*erreur* et la notion d'*incapacité*. L'écart entre le comportement de l'opérateur et ce qu'il aurait dû être, dépassant des limites d'acceptabilité dans des conditions données, est appelé une *erreur* humaine. Lorsque les conditions données de réalisation ne sont pas réunies, l'écart entre le comportement de l'opérateur et ce qu'il aurait dû être est qualifié d'*incapacité*. Ainsi, du point de vue des concepteurs, l'humain est perçu comme une potentielle entrave à la sûreté de fonctionnement du système.

Pendant un temps, les concepteurs ont donc tenté de limiter l'intervention de l'humain dans le pilotage du *système* au strict "nécessaire" (ce qui n'était pas automatisable). Ainsi, son rôle était d'intervenir uniquement en cas de "problème". Mais force a été de constater que ce n'était pas une solution. En effet, en reléguant l'activité de l'humain à une seule intervention en cas de situation imprévue, on l'a écarté de son implication au sein du système. Son comportement devenant encore plus imprévisible lorsque des décisions devaient être prises, notamment dû à une méconnaissance de l'état du *système*, de son historique, voir de la situation. L'opérateur étant ainsi dans l'incapacité de remplir son rôle, il ne pouvait plus permettre au système d'être résilient. Une défaillance humaine s'explique donc par une mauvaise adéquation entre les exigences de la tâche, et l'état et les capacités de l'opérateur [Kostenko, 2017].

La prise en compte de l'opérateur humain, au travers d'une vision technocentrée, en le considérant uniquement comme un simple utilisateur d'un système technique, a ainsi montré ses limites. On a donc par la suite tâché d'intégrer l'humain dans la conception, en tant que composant du système, notamment en termes de coopération homme-machine et d'interaction homme-machine. En conséquence, le système *à faire* ne peut plus être appréhendé au travers d'une unique dimension technique.

1.2.3.1 La place de l'humain dans le système

Garantir la sûreté de fonctionnement dans la conduite des procédés industriels est une tâche complexe. Comme on ne peut pas tout prévoir, ni tout automatiser, l'humain est censé pouvoir/devoir intervenir pour "aider" le système. Ainsi, « la conduite nécessite de maîtriser les problématiques rencontrées par les opérateurs, en situation dynamique, en tenant compte du facteur humain et des mécanismes de compréhension et de représentation mis en oeuvre » [Niel and Craye, 2002, p. 25].

La prise en compte du facteur humain est néanmoins beaucoup plus difficile à appréhender que la partie "machine" par les concepteurs. En effet, l'opérateur humain est perçu comme peu fiable (on ne lui fait pas confiance). Bien que la "machine" soit prévue pour une certaine utilisation, l'humain ne va pas forcément l'utiliser comme cela avait été prévu, pour plusieurs raisons.

- L'humain n'est pas un objet dont on peut fixer ou programmer le comportement (de façon déterministe). Cet indéterminisme dans le comportement de l'opérateur peut ne pas avoir été pris en compte, dans les représentations utilisées lors de la conception de la "machine".
- L'utilisation telle qu'elle a été prévue n'est pas toujours en adéquation avec la situation rencontrée par l'opérateur. L'activité de l'opérateur est située, il peut donc y avoir un écart avec la tâche prescrite et la tâche réelle de l'opérateur.

Il s'avère que l'humain n'est pas un objet technique dont on détermine le comportement lors de sa conception. Non ! L'humain est une personne qui pense, qui se construit et qui évolue : l'être humain est un système vivant. L'humain a ses propres intentions d'une part et il s'approprie les artefacts [Rabardel, 1995, Rabardel, 2005]. Ainsi, dans son activité, l'opérateur n'effectue pas uniquement les tâches qui lui sont confiées (activité productive), il cherche aussi à améliorer ses conditions de travail et les moyens pour réaliser son activité future (activité constructive) [Folcher and Rabardel, 2004]. L'opérateur réalise donc une activité productive (à court terme) et une activité constructive (à moyen et long terme). L'activité de l'opérateur est ainsi médiatisée et située. D'autre part, l'humain dans son activité doit prendre en compte des aspects organisationnels et sociologiques car son environnement de travail n'est pas uniquement constitué du système technique. En conséquence, l'environnement de travail de l'humain est un système sociotechnique.

Aujourd'hui, l'humain n'est plus considéré comme une simple entité réalisant une tâche prescrite. Il est considéré comme faisant partie du système, en tant que système autonome, ayant ainsi sa volonté propre. La place de l'homme dans le système a par conséquent évolué.

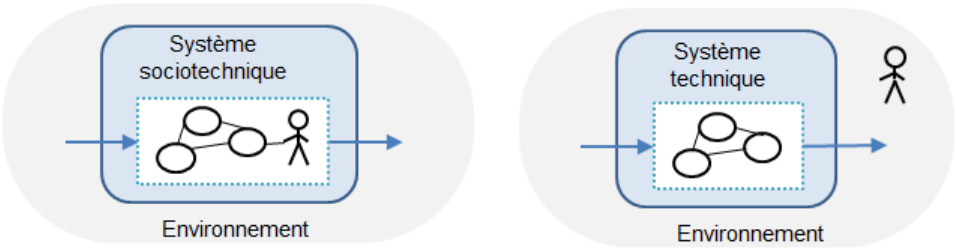


FIGURE 1.9 – La place de l'humain dans un système

Lors de la conception, le système de conduite du procédé est considéré comme un organisme ayant une dimension technique mais aussi une dimension sociologique : l'humain n'est plus considéré comme faisant partie de l'environnement du système mais comme faisant partie du système (Figure 1.9). Ainsi, la conception d'un système de conduite ne peut plus être appréhendée par une vision purement technocentrée, puisqu'il faut prendre en compte la dimension sociologique. Cette dimension est d'autant plus importante que l'humain est un système à l'intérieur du système (notion de système de système). La prise en compte d'une vision anthropocentrée est donc nécessaire, il ne s'agit plus de concevoir un système technique mais un système sociotechnique (Figure 1.10).

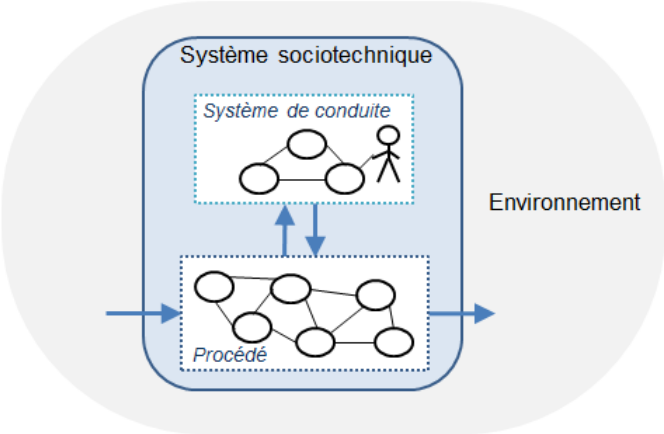


FIGURE 1.10 – La place de l'homme dans le système

1.2.3.2 Ingénierie des systèmes sociotechniques

Un système sociotechnique comporte ainsi trois composantes : une composante humaine (les êtres humains interagissant avec le système technique), une composante technique (les parties logicielles et matérielles du système) et une composante organisationnelle (des règles, procédures, normes ou standards qui encadrent les relations entre opérateurs, et entre opérateurs et machines) [Kostenko, 2017]. Un système de conduite de procédé peut néanmoins être vu comme un type particulier de système sociotechnique : un système sociotechnique *ouvert*.

Définition 1.7 (Système sociotechnique ouvert [Baxter and Sommerville, 2011])
Un système sociotechnique *ouvert* est un système qui s'adapte et qui poursuit des objectifs dans des environnements (externes).

Les objectifs du système peuvent être atteints de plusieurs façons différentes, cela implique donc l'existence d'un choix.

D'autre part, le système a des parties interdépendantes. L'environnement *interne* du système est composé de sous-systèmes techniques et sociologiques qui sont distincts mais interdépendants.

La performance du système repose ainsi sur l'optimisation conjointe des sous-systèmes techniques et sociologiques.

En effet, le système de conduite d'un procédé est ce qui permet d'adapter le fonctionnement (comportement) du procédé selon les évolutions de l'environnement. Comme nous l'avons vu, pour être en mesure de faire face à des aléas, lors de la conception, des redondances sont introduites dans le procédé. Le système de conduite dispose alors de plusieurs moyens pour atteindre les objectifs. D'autre part, la relation entre l'Homme et la Machine est un point important pour garantir le bon fonctionnement du système, en particulier en cas d'imprévu. La performance du système repose ainsi, entre autres, sur une optimisation de la relation entre l'humain et la partie technique, et ce, à différents niveaux d'abstractions (interaction Homme-Machine, coopération Homme-Machine, et activité médiatisée [Folcher and Rabardel, 2004]).

Lors de la conception d'un système de conduite, il est nécessaire d'être en mesure d'articuler une vision technocentrée avec une vision anthropocentrée. Il faut ainsi faire appel à l'ingénierie des systèmes sociotechniques. L'Ingénierie des Systèmes Sociotechniques (STSE) concerne l'utilisation systématique et constructive des principes et méthodes sociotechniques dans la spécification, la conception, le test, l'évaluation, en exploitation et dans l'évolution de système complexes à logiciel prépondérant [Baxter and Sommerville, 2011]. Afin d'optimiser conjointement sous-systèmes techniques et "humains", l'utilisateur final [6] doit être intégré dans les activités de conception du système. L'analyse de l'activité des opérateurs (utilisateur final) doit donc être prise en compte lors de la conception du système de conduite. Ainsi, l'anticipation de l'activité future (fonctionnement de l'opérateur et de son activité) fait partie intégrante du processus de conception [Béguin and Cerf, 2004].

L'introduction d'une vision anthropocentrée dans les approches de conception passe en partie par l'utilisation de représentions du fonctionnement de l'opérateur et de son activité (modèles de tâches, modèles d'activités, modèles cognitifs ...). Cela permet ainsi aux concepteurs de mieux appréhender la composante humaine du système. En effet, en ayant une meilleure représentation (et connaissance) du comportement de l'humain lors de l'exploitation du système, cela permet une meilleure prise en compte de la relation entre l'homme et la machine (en termes d'interactions Homme-Machine ou de coopération Homme-Machine, par exemple). Cependant, il est bien évident que la seule prise en compte de l'opérateur au travers de représentations n'est pas suffisante.

6. L'expression générique "utilisateur final" désigne les individus du sous-système "humain".

Il est ainsi nécessaire, lors de la conception du dispositif technique, de prendre en compte l'activité réelle de l'opérateur. Il faut alors s'intéresser à l'usage de ce futur dispositif, du point de vue de l'opérateur (l'utilisateur final). Afin d'adopter une vision anthropocentrée, l'utilisateur final doit participer aux activités de conception. On retrouve ainsi différents degrés de participation (information, consultation, prise de décision) [Darses and Reuzeau, 2004], selon l'implication de l'utilisateur dans les activités de conception. Le degré le plus faible de participation est celui d'*information*. On informe les opérateurs des décisions prises, et on collecte des informations relatives aux opérateurs ainsi que sur leurs expériences utilisateur. Un deuxième degré de participation est celui de *consultation*, où les avis et suggestions des utilisateurs sont recueillis par rapport à des choix de conception. Enfin, le dernier degré de participation est celui de *prise de décision*, où l'utilisateur final est considéré comme un acteur de la conception. L'utilisateur a donc dans ce cas un pouvoir de décision, et participe à la conception du dispositif technique.

Ainsi, dans le cadre d'une approche de conception dite "centrée sur l'utilisateur", l'utilisateur final est généralement impliqué lors de phases d'évaluation et de tests d'utilisabilité [Darses and Reuzeau, 2004]. L'utilisateur final dans ce type d'approche est consulté, mais n'a cependant pas de réel pouvoir de décision. En effet, dans ce type d'approche, contrairement à une "véritable" approche de conception participative, on ne fait pas intervenir l'utilisateur final dans le cycle de conception en tant qu'acteur (de celle-ci), il n'est donc généralement pas associé à la rédaction des spécifications [Darses and Reuzeau, 2004].

La conception participative, en intégrant l'utilisateur comme un acteur de la conception (ayant un pouvoir de décision), vise à obtenir une meilleure expression du besoin et des exigences par rapport à l'usage qui sera fait du futur dispositif [Darses and Reuzeau, 2004]. Pour cela, l'utilisateur final se place dans une activité constructive : il participe à l'élaboration des outils de son activité future. En intégrant l'opérateur en tant qu'acteur de la conception, cela permet, d'une part, de mieux prendre en compte l'activité réelle (en situation dynamique), et d'autre part, cela permet de commencer plus tôt le processus d'appropriation de la part de l'opérateur. Ce dernier point est d'autant plus important dans le cadre de conception de systèmes de conduite de procédés, puisqu'il faut en outre éviter des défaillances humaines (aspect sûreté de fonctionnement et résilience du système).

Pour cela, il faut, entre autre, être en mesure de garantir l'adéquation entre la tâche et les capacités de l'opérateur, et ce, à plus forte raison du fait du caractère résilient du système conduite. En effet, la résilience, selon [Ruault et al., 2009], « concerne la capacité d'un système sociotechnique à s'ajuster face à des événements perturbateurs, à s'y adapter, et à apprendre les règles d'adaptation adéquates, quand les perturbations sont en dehors du périmètre spécifié des mécanismes d'adaptation du système, c'est-à dire des menaces irrégulières et des menaces sans précédent. En outre, la résilience comprend aussi la détection de l'atteinte des limites des capacités d'adaptation, en fonction de la configuration du système sociotechnique ». Il y a donc un réel besoin de mise en place de tests du système, en situation dynamique, pour que l'opérateur puisse évaluer la solution, proposer des améliorations, et potentiellement exprimer de nouveaux besoins. Il faut ainsi disposer de prototypes tout au long de la conception pour

pouvoir effectuer des tests utilisateurs.

1.2.4 Mise en place de tests

Afin de vérifier et valider le système de conduite durant la conception, différentes techniques sont utilisées pour vérifier les propriétés du système.

Définition 1.8 (Propriété [Chapurlat, 2007])
Une propriété est une caractéristique intrinsèque (comportementale, fonctionnelle, structurelle ou organique, dépendante ou non du temps) d'une entité (un système, un modèle, une entité de modélisation, un phénomène etc.)

La vérification (au sens large) peut se faire au moyen de preuves formelles (situation statique) ou bien via la mise en place de tests (situation dynamique). Face à l'importance des propriétés du système (sociotechnique ouvert), concernant son caractère adaptable en situation dynamique, un intérêt particulier est porté sur les approches de tests. Le test du système de conduite (dans son ensemble) nécessitant de pouvoir tester le système de contrôle-commande, nous nous intéresserons en particulier à celui-ci dans la suite de la section.

Historiquement, le test d'un système de commande se faisait en utilisant une maquette (physique) du système à commander, voire, à défaut, directement sur le système réel. Le test sur une maquette, bien qu'il puisse en théorie être effectué plus tôt qu'en phase de tests réels, arrive relativement tard dans le cycle de conception. En effet, la maquette n'est pas forcément disponible ou existante. Même lorsqu'une maquette est disponible pour effectuer les tests, la complexité des systèmes ayant augmenté, la maquette ne représente en général qu'une partie du système. Ainsi, comme le souligne [Schludermann et al., 2000], les tests sont incomplets puisqu'il manque l'interaction avec le reste du système, et donc une grande partie des tests et du debugg se fait sur site (sur le système réel). Comme il est très difficile de tester un système de commande (partie matérielle et logicielle) avant l'implémentation et le couplage avec le procédé à commander, les programmes de commande ont souvent été terminés sur place [Auinger et al., 1999]. En plus des coûts engendrés par ce type d'approche, il peut être dangereux de tester le comportement dans des situations critiques (potentiels dégâts sur le matériel et/ou mise en danger de vies humaines) [Schludermann et al., 2000]. D'autre part, certaines conditions de tests peuvent être difficiles à reproduire.

Afin de pouvoir effectuer les tests plus tôt et en toute sécurité, l'idée a donc été de remplacer la maquette (physique) du système par une maquette virtuelle (simulation du procédé) [Riera et al., 2009]. Ce nouveau type d'approche de test a été désigné via le terme d'*émulation* puis de *soft-commissioning*. Selon [Schiess, 2001], cette approche est née du mariage de deux disciplines, la simulation et la conception de commande, dans le but d'accomplir effectivement les opérations du système dans un monde virtuel. Ainsi, au lieu d'utiliser le système réel à commander, on utilise un modèle de simulation qui imite le comportement du système à commander (réel) auquel on connecte le système de commande [Pfeiffer et al., 2003]. L'objectif étant de pouvoir tester le fonctionnement de systèmes de commande (réel) en les connectetant à des modèles de simulation [Johnstone et al., 2007], cette approche est un outil

(puissant) qui permet de tester et debugger le code de commande avant l'implémentation sur site [Schiess, 2001].

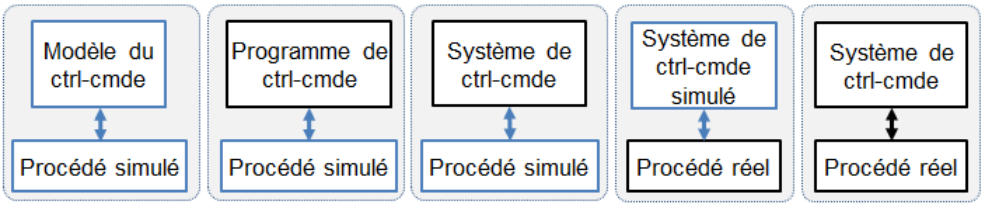


FIGURE 1.11 – Les différentes approches de test du système de contrôle-commande

L'utilisation d'une maquette virtuelle offre différentes possibilités pour assister les activités de conception : on peut tester le modèle de la commande, le programme de commande réel et le système de contrôle-commande complet (avec l'automate programmable industriel réel). Les différentes approches de tests possibles sont synthétisées sur la Figure 1.11. D'autre part, il est aussi possible d'avoir une maquette physique du procédé et de le connecter à un système de contrôle-commande simulé, afin d'obtenir un prototype de ce système.

Bien que ces approches s'inscrivent dans une vision technocentrée, elles permettent également de pouvoir conduire des tests à des fins de validation. En effet, en disposant d'un prototype virtuel de la partie technique du système de conduite, tôt dans le cycle de conception, cela permet de pouvoir effectuer des tests avec l'utilisateur final. Les tests peuvent ainsi être conduit par rapport à l'interface Homme-Machine lorsque l'on s'intéresse en particulier à l'interaction entre l'opérateur et la machine, ou bien en termes de coopération Homme-Machine lorsque l'on s'intéresse au système de conduite dans son ensemble.

1.2.5 Bilan

Comme nous avons pu le voir, un système de conduite de procédé est un système ayant un degré important d'automatisation. Il permet de commander, surveiller et superviser le fonctionnement du procédé. Le système de conduite est ainsi ce qui permet d'adapter le fonctionnement du procédé aux évolutions de l'environnement, qu'elles soient prévues ou non. Pour ce faire, il dispose d'une structure hiérarchisée dans laquelle l'humain est responsable de la conduite du procédé.

La conception du système de conduite est une tâche complexe, il faut non seulement garantir le bon fonctionnement du système dans des situations normales, mais aussi s'assurer que le système pourra faire face lors de situations inattendues. Pour faire face aux évolutions de l'environnement, et garantir une sûreté de fonctionnement, le système dispose de plusieurs moyens pour atteindre ses objectifs. Il dispose également de mécanismes pour pouvoir s'adapter, dont le rôle de l'opérateur humain devient prépondérant lorsque l'on sort des conditions anticipées d'exploitation. L'opérateur humain n'est donc pas un simple utilisateur du système de conduite, il est une composante du système. A ce titre, éviter l'apparition de défaillances du système de conduite nécessite de prendre en compte la relation entre l'Homme et la Machine lors de la conception.

Ainsi, un système de conduite d'un procédé peut être appréhendé via la notion de **système sociotechnique ouvert**. Cette notion met en évidence le fait que le système poursuit des objectifs (qu'il peut atteindre de différentes manières) dans un contexte dynamique auquel il s'adapte (aspect de sûreté de fonctionnement et de résilience). Elle met aussi en évidence le fait que les sous-systèmes techniques et sociologiques qui le composent sont distincts mais interdépendants. La performance du système repose ainsi sur une optimisation conjointe des ces sous-systèmes. La conception d'un système de conduite se place donc dans le cadre de la conception de systèmes sociotechniques. Ceci nécessite une articulation entre vision technocentrée et anthropocentrée, notamment au travers d'une approche de **conception participative**. Le principe de cette approche de conception est d'intégrer l'utilisateur final (l'opérateur humain) en tant qu'acteur de la conception. Il s'agit donc de le faire intervenir dans les activités de conception, de la spécification à l'évaluation. Pour cela, des cycles de conception courts et itératifs sont nécessaires, et il faut entre autres pouvoir disposer de prototypes de solution très tôt. En effet, la mise en oeuvre de la conception participative passe en partie par la réalisation de tests utilisateurs tout au long de la conception.

D'autre part, la performance du système reposant sur une optimisation conjointe des sous-systèmes techniques et sociologiques, il s'agit aussi de tester le système d'un point de vue technique (le vérifier). Afin d'être en mesure de vérifier le système de conduite au plus tôt, différentes approches de tests peuvent être utilisées pendant le cycle de conception. Disposer de prototypes virtuels du système de conduite et du procédé, tôt dans la conception, permettrait, après une mise en oeuvre de tests de vérification, la mise en place de tests de validation (tests utilisateurs). Ceci permettrait, de plus, d'intégrer des vérifications et des validations tout au long des processus de conception des systèmes.

Dans cette section, nous nous sommes ainsi intéressés aux problématiques de conception de systèmes de conduite, raffinant ainsi le contexte de nos travaux. Il en est ressorti, entre autres, un besoin de vérification et validation tôt dans la conception via la mise en oeuvre de tests. La section suivante s'intéressera à un cas industriel : la conception de système de conduite de navire.

1.3 Contexte industriel : système de conduite de navire

D'un point de vue industriel, la conception d'un système se place dans un contexte de concurrence généralisée : les industriels cherchent ainsi à réduire les coûts et délais de conception. Traditionnellement, dans les projets de conception de grands systèmes sociotechniques (par exemple un navire), le projet de conception du procédé et celui du système de conduite sont articulés de manière séquentielle. C'est à dire qu'une fois que toute la partie "physique" du navire est conçue (figée), on passe à la conception de la partie logicielle (*Programmation* sur la Figure 1.12), c'est à dire le système de conduite. Or, comme évoqué précédemment, il est nécessaire de prendre en compte les besoins du système de conduite au plus tôt. Une approche de conception simultanée doit donc être mise en oeuvre ainsi qu'une conception participative pour prendre en compte la dimension humaine.

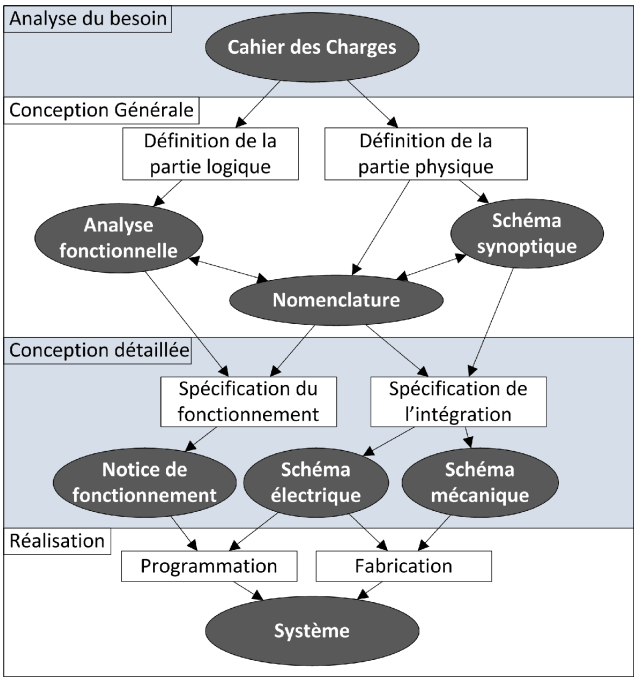


FIGURE 1.12 – Processus de conception générique "simplifié"proposé par [Bignon et al., 2010]

SEGULA Technologies a donc cherché à améliorer ses méthodes et outils de conception, notamment dans le cadre de la conception de système de conduite de navire (cf Figure 1.13). Une démarche outillée a été proposée, afin de permettre la conception de la partie logicielle dès le début de la conception du navire et pour réduire les efforts de communication au sein d'équipes pluridisciplinaires. La conception reposant sur l'utilisation de modèles métiers, le but est de créer des passerelles entre les différents modèles utilisés par les concepteurs, pour leur permettre de se concentrer sur leur coeur de métier. Ainsi, l'intérêt de cette démarche est de faciliter les échanges entre concepteurs et d'assurer la cohérence entre les modèles grâce à l'IDM.

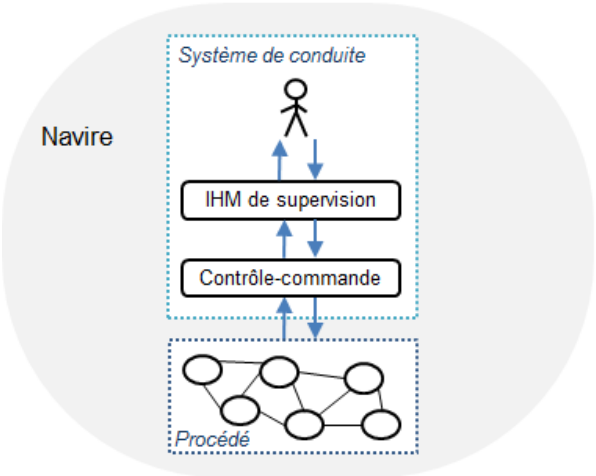


FIGURE 1.13 – Le système de conduite dans un navire

1.3.1 La démarche outillée de conception *Anaxagore*

Les travaux précédemment menés [Bignon, 2012, Goubali et al., 2014] ont abouti à la création d'un outil d'aide à la conception générant conjointement programme de commande et interface de supervision. Cet outil, nommé *Anaxagore*, utilise les techniques de transformations de modèles de l'IDM pour générer la commande et l'interface de supervision à partir d'un modèle métier (fourni par un mécanicien) et d'une bibliothèque d'éléments. Le modèle métier est un schéma P&ID (*Piping & Instrumentation Diagram*), basé sur la norme [ANSI/ISA-5.1, 1992], décrivant le système physique. Un élément est défini comme l'unité constitutive du procédé du système. Il peut être relatif à du matériel (vanne, pompe ...) ou à des fonctionnalités du système. Chaque élément de la bibliothèque contient plusieurs *vues*. Le concept de *vue* est ici une extension de celui utilisé par Lallican [Lallican et al., 2007]. Il correspond aux différents points de vue des concepteurs. Un élément de la bibliothèque contient ainsi plusieurs vues dont celle concernant la supervision, et celle concernant la commande.

Un autre modèle métier est pris en compte pour générer l'interface de supervision suivant des critères ergonomiques [Rechard, 2015] : l'analyse du domaine de travail. Ce modèle permet de représenter l'ensemble des possibilités (par rapport aux ressources et aux fonctions) pouvant être mis en oeuvre pour atteindre les objectifs du système. Cela s'apparente à une analyse fonctionnelle dans laquelle sont synthétisés les objectifs, les critères pour atteindre ces objectifs, les fonctions à mettre en oeuvre, ainsi que les moyens disponibles permettant de réaliser ces fonctions.

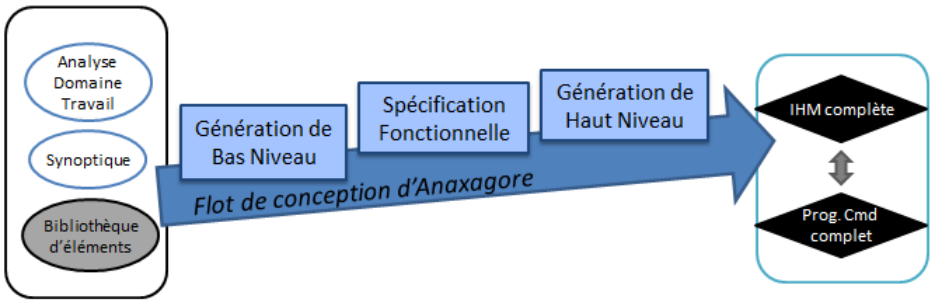


FIGURE 1.14 – Flot de conception d'Anaxagore simplifié

Le flot de conception, permettant de générer IHM de supervision et programme de commande, peut être décomposé en trois phases (Figure 1.14). **La première phase** consiste à générer l'interface de supervision et le programme de commande associé de bas niveau (*Génération de Bas Niveau*, Figure 1.14). La chaîne de contrôle-commande générée permet uniquement de commander les éléments du système un à un, via les **commandes élémentaires**. Une vanne pourra, par exemple avoir deux commandes élémentaires : une correspondant à sa commande d'ouverture, l'autre à sa commande de fermeture. **La deuxième phase** correspond à la *spécification fonctionnelle* des commandes globales (aussi appelées commandes de haut niveau) [Goubali, 2017]. Pour spécifier ces commandes globales, les techniques de la programmation par l'utilisateur final sont utilisées. L'intérêt est de permettre à l'expert mécanicien de créer directement les programmes de haut niveau sans avoir recours à des connaissances informatiques. Pour ce faire, une interface de spécification (basé sur l'interface de bas

niveau) permet d'enregistrer des séquences de commandes élémentaires. L'expert enregistre deux exemples permettant de réaliser la fonction à spécifier. Les points de départ et d'arrivée sont différents d'un exemple à l'autre. A partir des exemples enregistrés, un module permet de trouver toutes les configurations associées à la fonction qui est en train d'être spécifiée (généralisation). Ensuite, une phase de rejeu permet de rejouer les différentes configurations trouvées afin que l'expert mécanicien puisse vérifier que cela correspond à ce qu'il a voulu spécifier. Une fois que toutes les commandes globales ont été spécifiées et validées, on passe à **la dernière phase** (la *Génération de Haut Niveau*, Figure 1.14). On y génère l'interface de supervision et les commandes de haut niveau, ainsi que la partie qui gère la reconfiguration. On obtient ainsi le système de contrôle-commande complet (d'un point de vue logiciel).

Toutefois, avant d'exécuter le flot de conception, les modèles d'entrée doivent être vérifiés, notamment via l'utilisation de méthodes formelles. Ainsi, les travaux de [Mesli et al., 2016] et [Mesli-Kesraoui et al., 2016b] permettent de vérifier la bibliothèque d'éléments, tandis que ceux de [Mesli-Kesraoui et al., 2016a] permettent la vérification du schéma P&ID (synoptique). La vérification du modèle de l'analyse du domaine de travail, pour la prise en compte de critères ergonomiques de l'IHM, peut quant à elle être faite à travers la méthode TMTA [Rechard et al., 2015].

1.3.2 Bilan/Manque par rapport aux besoins

La démarche outillée *Anaxagore*, se place dans un contexte de conception simultanée et vise la mise en place d'une conception centrée sur l'utilisateur final (conception participative). L'outil Anaxagore permet de créer une passerelle entre les concepteurs via l'utilisation de *transformations de modèles*, optimisant ainsi les efforts de communication des différents concepteurs. D'autre part, il permet de *limiter l'introduction d'erreurs d'implémentation* grâce à l'utilisation de la *génération automatique*. La *cohérence syntaxique* entre le code de contrôle-commande et l'interface de supervision y est également assurée, grâce à la génération conjointe. De plus, cet outil est adapté au *processus itératif de conception*. En effet, le gain de temps noté lors de la génération du code et de l'interface permet d'envisager l'obtention d'alternatives relativement rapidement, ainsi qu'une rapide prise en compte des modifications des données d'entrée, en propageant ces modifications.

Afin de s'assurer de la qualité des programmes générés, des techniques de vérification par méthodes formelles permettent de vérifier et valider les modèles d'entrée utilisés pour la génération conjointe de l'IHM et de la commande. Cependant, la démarche actuelle n'intègre pas de techniques de vérification dynamique. Il faut donc attendre de disposer d'une maquette ou d'un prototype physique pour effectuer des tests (Figure 1.15). Par conséquent, les tests ne peuvent être effectués que dans des phases tardives de la conception.

D'autre part, la prise en compte de l'humain (utilisateur final) se fait uniquement au travers de l'introduction de critères ergonomiques lors de la génération de l'IHM. Afin de s'inscrire dans une démarche de conception centrée sur l'utilisateur (conception participative), il faudrait être en mesure de pouvoir effectuer des tests utilisateurs dès le début des phases

de conception. Disposer d'un prototype virtuel du procédé (Figure 1.15), faciliterait la mise en place des validations par les utilisateurs finaux.

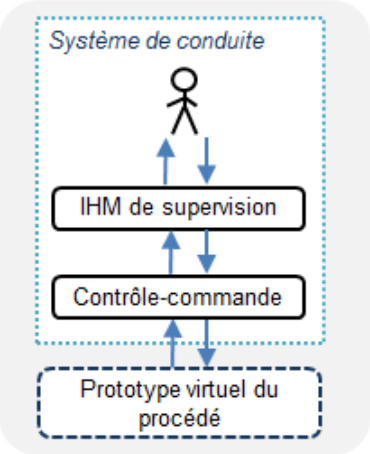


FIGURE 1.15 – Besoin d'un prototype pour effectuer des vérifications/validations

En disposant, d'un prototype virtuel du procédé, la chaîne de contrôle-commande pourrait être testée par rapport aux aspects : *aide à la décision et à la surveillance* en s'intéressant par exemple au système d'alarmes, *fonctions de haut niveau*, et *adaptation du système face aux aléas*. Concernant *l'aide à la décision et à la surveillance*, il peut s'agir de répondre aux questions suivantes :

- Les alarmes se déclenchent-elles au bon moment ?
- Les alarmes s'affichent-elles "correctement" au niveau de l'IHM ?
- La cohérence sémantique du système d'alarmes est-elle assurée entre la partie de contrôle-commande et l'affichage sur l'IHM de supervision ?

Les questions concernant les *fonctions de haut niveau* peuvent être :

- Les fonctions de haut niveau correspondent-elles aux besoins des utilisateurs ?
- Les commandes de haut niveau associées aux fonctions de haut niveau permettent-elles de délivrer les différents services pour lesquels elles ont été conçues ?
- Les fonctions de haut niveau prennent-elles en compte des contraintes de sûreté de fonctionnement (priorités entre plus plusieurs fonctions, fonctions qui ne doivent pas être mises en oeuvre simultanément ..) ?

Enfin, en présence d'aléas, que ce soit la défaillance d'une ressource, un changement d'objectif (et donc de services à fournir) ou bien un changement de situation (c'est à dire du contexte, de l'environnement), les questions concernant l' *adaptation du système* peuvent être :

- Le système est-il capable de continuer à délivrer le(s) service(s) requis par la situation ?
- Lorsque le système se reconfigure (changement de son organisation interne), la transition entre les deux respecte-t-elle les contraintes de fonctionnement et de sécurité ?
- Lorsque plusieurs fonctions ne peuvent/doivent pas être mises en oeuvre simultanément, le système est-il capable de gérer la priorité entre les fonctions ?

Ainsi, l'ajout de techniques de vérification par simulation du procédé permettrait de garantir la qualité des programmes générés au plus tôt, en termes de cohérence sémantique et de prise en compte de la sûreté de fonctionnement. D'autre part, cela permettrait de pouvoir

mettre en place des tests utilisateurs pour valider la chaîne de contrôle-commande, et ainsi se placer réellement dans un contexte de conception participative.

1.4 Un besoin de vérification/validation au plus tôt

Outre le contexte industriel de concurrence généralisée, la conception des systèmes de conduite de procédés s'inscrit dans le cadre de conception de **systèmes sociotechniques ouverts**. Afin de s'assurer que les exigences et spécifications du système *à faire* sont complètes (par rapport au client et aux futurs utilisateurs), les cycles de conception doivent ainsi être courts, itératifs et inclure l'utilisateur final comme un acteur de la conception. Il est de plus nécessaire d'intégrer des vérifications et validations tout au long de la conception, en particulier pour optimiser conjointement les sous-systèmes techniques et humains. Il ressort ainsi deux besoins :

- Un besoin **de vérifier** (par rapport aux spécifications) le système de conduite, que ce soit au niveau de l'interaction homme-machine ou du système homme-machine.
- Et un besoin **de validation** (par l'utilisateur final) que le système de conduite répond au besoin, en termes d'interaction Homme-Machine, de coopération Homme-Machine et d'activité médiatisée.

La mise en place de ces vérifications et validations requiert toutefois un prototype virtuel du procédé (Figure 1.16), pour effectuer les tests en toute sécurité (malgré l'introduction d'aléas) d'une part, et pour pouvoir les réaliser le plut tôt possible.

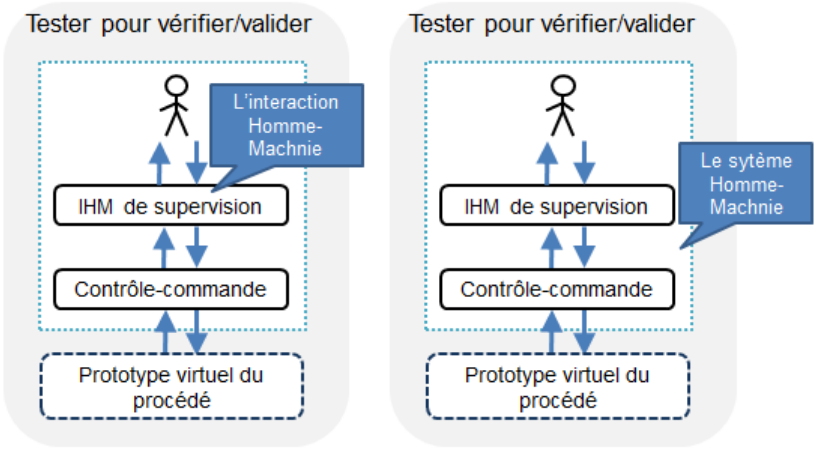


FIGURE 1.16 – Effectuer des vérifications/validations

Cependant, dans un contexte industriel de concurrence généralisée, où les coûts et les délais doivent être réduits, l'intégration de vérifications et validations peut paraître en contradiction avec les contraintes économiques. En ce plaçant dans un **contexte d'ingénierie basée sur les modèles**, et en utilisant des techniques de transformation de modèles de l'IDM, le temps et les coûts nécessaires à l'obtention d'un prototype virtuel ne devraient pas augmenter de manière disproportionnée. Au contraire, le temps "gagné" devrait permettre de pouvoir mettre en place les vérifications et validations. Plus tôt le système de conduite sera testé, plus tôt les

erreurs de conception seront détectées et corrigées, réduisant ainsi les coûts et les temps de (re-)conception.

Dans le cadre d'approche de **conception participative**, il est essentiel de pouvoir disposer de prototypes du système *à faire* très tôt dans la conception, afin de pouvoir confronter l'utilisateur final aux solutions envisagées, et potentiellement modifier les spécifications. Une approche d'**ingénierie simultanée** de la conception du système de conduite et du procédé va dans ce sens, les activités avançant en parallèle, l'obtention d'un prototype peut être disponible plus tôt. Cependant, pour intégrer les tests le plus tôt possible, on ne peut pas attendre de disposer d'un prototype physique, l'obtention de prototypes virtuels semble donc être une solution.

L'obtention de prototypes virtuels, passe par l'utilisation de techniques de simulation, il convient donc de s'intéresser à l'apport de la simulation pour la mise en place de vérifications et validations. Ceci fera l'objet du chapitre suivant.

2

Apport de la simulation

Sommaire

Face aux besoins évoqués précédemment, l'utilisation de la simulation semble être un moyen d'y répondre. En effet, la simulation dans les sciences, contrairement au joueur de football qui "simulerait" une blessure, ne consiste pas à faire semblant (ou à imiter) quelque chose *afin de duper quelqu'un*. Non, dans les sciences, la simulation sert à conduire des expérimentations (pour répondre à des questions lorsque le système réel n'est pas disponible ou pas accessible), ou bien, elle sert à acquérir de l'expérience (pour se former ou s'entraîner, comme par exemple sur un simulateur de vol).

Définition 2.1 (Simulation [Ören and Waite, 2010])
Simulation is performing goal-directed **experiments** using a model of a dynamic system. Simulation is gaining **experience**, by use of a representation of a system, to enhance any one of three types of skills :

- *motor skill* (by virtual simulation, or simulators),
- *decision making and communication skills* (by constructive simulation, gaming simulation, or by serious games),
- and *operational skills* (by live simulation).

Ainsi face aux problématiques de conception dans le cadre de la conduite de procédé, la simulation semble avoir le double avantage de permettre le test du système de contrôle-commande et la réalisation de tests utilisateurs. D'autre part, puisque la simulation peut aussi

servir à former du personnel, cela laisse entrevoir une possibilité de réutiliser la représentation du système à commander (permettant de tester le système de contrôle-commande), pour former les futurs utilisateurs du système de conduite.

Afin d'évoquer plus en détail les apports et intérêts que peut offrir la simulation, on s'intéressera à l'utilisation de la simulation en tant que support des activités d'ingénierie dans un premier temps. Un tour d'horizon des méthodes de test actuellement utilisées dans la conception de systèmes de commande sera notamment évoqué. Puis dans un second temps, on s'intéressera à l'élaboration d'une simulation.

2.1 La simulation : un support pour l'ingénierie

Depuis des siècles, modèles et simulations sont utilisés par les concepteurs/ingénieurs pour à la fois vérifier leurs propres pensées, et pour communiquer leurs concepts aux autres [INCOSE, 2011]. Il nous paraît intéressant, en guise d'introduction, de commencer par retracer brièvement l'histoire de la *simulation*.

Au XVI^{ème} siècle, la simulation était utilisée par les ingénieurs pour montrer les effets d'un phénomène à l'aide d'une maquette (c'est-à-dire un "modèle réduit"). La maquette servait ainsi à illustrer un argument ou à remplacer un calcul par une mesure directe sur celle-ci [Varenne, 2014]. Pour permettre des simulations, on utilisait donc un modèle *physique* dont le but était d'imiter les performances observables de l'objet/système modélisé. Cette représentation (physique) de la réalité ne visait pas à reproduire le fonctionnement (interne) de l'objet d'intérêt, mais la partie visible. Une approche de modélisation de type boîte noire était ainsi utilisée.

Au fil du temps, la notion de modèle a évolué, ce n'est plus "un construit physique réel à visée de reproduction simplifiée" (pour reprendre l'expression de [Varenne, 2014]), mais une aide pour raisonner via l'utilisation d'une représentation visuelle ou figurative. Ainsi, un modèle lorsqu'il n'est pas une représentation physique, s'appuie sur un langage (de modélisation). Avec les avancées technologiques, la notion de simulation a aussi évolué, elle est devenue une alternative pour calculer les évolutions d'un modèle (mathématique), symbolique. Au lieu d'une maquette, on a utilisé un modèle mathématique et la simulation est devenue du calcul numérique de ce modèle, pas à pas, en prenant des échantillons de temps infinitésimaux. Aujourd'hui, la simulation informatique est un moyen de représenter la dynamique d'un système, notamment pour pouvoir réaliser des expérimentations.

Ainsi, au cours du XX^{ème} siècle, deux significations techniques ont été rajoutées [Ören, 2007]. Dans l'une, la simulation sert à fournir de l'expérience (pour se former ou s'entraîner) dans des conditions que l'on contrôle. Dans l'autre, elle sert à réaliser des expérimentations (pour de l'aide à la décision, la compréhension ou l'éducation).

Aujourd'hui, la simulation peut être utilisée tout au long des phases de conception, de développement et même d'exploitation d'un système. Dans les phases de conception préliminaire (phase de conceptualisation), elle peut aider à évaluer la faisabilité d'un projet, ou aider

à comparer différentes solutions. Pendant la conception, la simulation permet d'améliorer la confiance dans la proposition d'une solution avant de la développer [Foures, 2015]. Elle permet aussi d'évaluer le respect des exigences, et d'envisager des améliorations de la solution. Dans les phases de développement, elle apporte un support, et permet d'aider à la validation du comportement du système à chaque étape. La simulation intervient aussi pour former le personnel, et comme aide lors de l'exploitation (aide au diagnostique, aide à la réparation...).

2.1.1 La simulation pour la vérification/validation

La simulation peut être utilisée afin d'effectuer les activités de V&V pendant tout le cycle de conception et de développement. *Les simulations peuvent refléter les fonctions d'un système ou bien la structure détaillée du système. Ces simulations sont composées des représentations des éléments du système, connectés de la même manière que dans le système réel* [INCOSE, 2011]. Généralement, la simulation est exécutée au travers de scénarios dans le domaine temporel pour simuler le comportement du système réel. L'exemple illustratif donné par l'INCOSE est celui d'un système de contrôle-commande d'un fluide. La simulation d'un tel système est constituée des représentations de la tuyauterie, de pompes, de vannes, de capteurs, d'un circuit de commande et du fluide qui circule à travers le système.

L'avantage de l'utilisation de la simulation est qu'elle est plus facile à mettre en oeuvre que des techniques de preuve d'une part, et d'autre part, elle permet l'analyse de système complexe plus facilement que les méthodes mathématiques/analytiques (en termes de temps par exemple). Cependant, elle ne peut pas être utilisée pour obtenir une preuve absolue, car le système est testé sur un certain nombre de scénarios, tous les cas ne sont donc pas abordés. De plus, le modèle de simulation est conçu selon un objectif et selon la manière dont est perçu le système. Ceci peut poser un problème au niveau des hypothèses, qui peuvent être erronées ou manquantes, par exemple.

Quel que soit le type de technique de simulation utilisé pour représenter la dynamique du système d'intérêt, de façon assez générale, on retrouve quatre types d'approches pour les plateformes de tests en fonction de la place de la simulation dans le cycle de conception [Foures, 2015] :

- En début de conception, il y a l'approche de simulation "*Model in the loop*" (MiL). On utilise le modèle décrivant le comportement du système à valider en le couplant avec un modèle de l'environnement, afin de constater si le modèle satisfait les exigences principales (du système).
- Ensuite, on trouve l'approche de simulation "*Software in the loop*" (SiL). Cette fois, on utilise le programme cible qui implémente le modèle (le programme final). On cherche ici à garantir la cohérence sémantique.
- Puis, apparaît l'approche de simulation "*Processor in the loop*" (PiL), parfois appelée "*Controller in the loop*". On s'intéresse ici à l'équivalence comportementale du programme après son intégration dans le processeur cible, cependant l'environnement reste simulé.
- Enfin, la dernière approche de simulation correspond à celle dite "*Hardware in the*

loop" (HiL). Cette fois-ci on utilise le contrôleur physique final (par exemple l'automate programmable) mais l'environnement est toujours simulé avec cependant des contraintes de temps réel.

L'approche Processor-in-the-loop, à notre niveau de granularité dans notre cadre d'application, semble se confondre avec une approche Hardware-in-the-loop. On ne considérera donc par la suite que trois types.

Dans le domaine de la mécatronique, [Bishop, 2005] a identifié différents types d'approche de simulation par rapport au temps de calcul. La première est sans considération de limite de temps (matériel). Elle sert notamment à la vérification de modèles théoriques, à avoir un aperçu basique du comportement, à la conception du procédé ou encore à la conception du système de contrôle. Ensuite, on trouve des approches de simulation dites "temps-réel" où les valeurs des signaux d'entrée-sortie ont une dépendance temporelle en correspondance avec la réalité (approche de type SiL et HiL par exemple).

Parmis les approches de tests des systèmes de contrôle-commande [Pfeiffer et al., 2003, Auinger et al., 1999, Versteegt and Verbraeck, 2002, Lee and Park, 2014], on retrouve :

- la simulation hors ligne, qui correspond à l'approche Model-in-the-loop,
- l'émulation et le virtual commissioning, qui correspondent à une approche Hardware-in-the-loop ou Software-in-the-loop (selon que l'automate programmable industriel cible est utilisé ou non).

Malgré des similitudes dans les approches de test par simulation, la terminologie différant d'un domaine à l'autre, une synthèse de ces différentes approches de test est représentée sur la Figure 2.1.

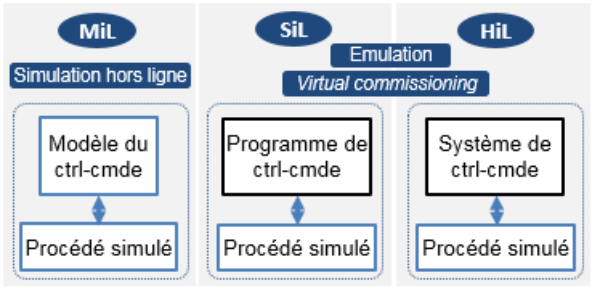


FIGURE 2.1 – Synthèse des approches de test

Dans le cadre de la conception d'un système de conduite de procédé, via une approche d'ingénierie simultanée et participative, les approches de test de type SiL et HiL semblent bien adaptées pour permettre des tests utilisateurs. La suite de la section s'intéressera ainsi aux approches de test par *émulation* et au *virtual commissioning*.

2.1.2 L'*émulation* et le *virtual commissioning*

2.1.2.1 Le concept d'*émulation*

L'objectif de l'*émulation* est de pouvoir tester le fonctionnement de systèmes de commande (réels) en les connectant à des modèles de simulation [Johnstone et al., 2007], avant

implémentation sur site. Si une approche par *émulation* consiste à utiliser des modèles de *simulation* à la place d'un prototype physique (ou du procédé réel), alors pourquoi parler d'*émulation* ?

Au regard des définitions dans la littérature, la distinction entre *émulation* et *simulation* est souvent ambiguë (malgré des efforts de clarification). Nous attribuons cela aux (différentes) évolutions des disciplines relativement récentes (l'électronique, l'informatique, la simulation, le contrôle-commande...). Rien que pour la simulation, il existe une centaine de définitions [Ören, 2011b] et près de 400 types de simulation [Ören, 2011a]. Il peut donc s'avérer difficile de s'y retrouver face à une littérature aussi riche.

Il semble que dans le génie automatique, on ait voulu différencier la notion de *simulation* de celle *d'émulation*. Mc Gregor a notamment introduit la notion de modèle d'*émulation* [McGregor, 2002] afin de se différencier du modèle utilisé avec ce qu'il appelait un "pur" modèle de simulation.

Définition 2.2 (Modèle d'émulation, selon [McGregor, 2002])
Un modèle d'*émulation* est celui où des parties fonctionnelles du modèle sont réalisées par une partie du système réel.
Un modèle d'*émulation* est celui où une partie du système réel est remplacée par un modèle.

Une explication (ou des éléments d'explication) de cette distinction entre émulation et simulation, se retrouve dans la définition donnée par Pannequin. Pour lui, c'est une différence de finalité car « la finalité de la simulation est d'observer l'évolution des états internes du modèle placé dans une situation prédéfinie » alors que la finalité de l'émulation, c'est de « reproduire l'interaction du système à représenter avec son environnement » [Pannequin, 2007]. De plus, il considère que contrairement à un modèle de simulation, le but d'un modèle d'émulation est de « reproduire la réponse du système réel à des séquences d'entrées, afin d'être utilisé dans un système plus vaste ».

La différence entre *simulation* et *émulation* semble donc être liée à la définition de la simulation et à son usage. La *simulation* semble avoir été perçue comme une technique d'analyse d'un objet d'étude, sans toutefois tenir compte du système plus vaste dans lequel il est placé. Or pour tester un système de contrôle-commande, il est nécessaire de prendre en compte le système (complexe) dans son ensemble, et par conséquent de reproduire les interactions du système de contrôle-commande avec son environnement.

D'autre part, pour vérifier le programme de contrôle-commande, il ne s'agit pas de juste reproduire des entrées-sorties, il faut aussi s'intéresser au comportement du système qui est commandé. En effet, comment savoir si le contrôle-commande restreint correctement les évolutions possibles du système qui est commandé, si l'on ne s'intéresse pas à son état ? Il semblerait donc que ce qui doit être "émulé" pour pouvoir tester le programme de contrôle-commande, n'est pas le système qui doit être commandé mais la plateforme cible d'exécution du programme.

Dans [Ören, 2007], la distinction entre simulation et émulation vient du fait que dans le cas de l'*émulation*, on utilise un système à la place d'un autre système. Ainsi afin de représenter

un système, au lieu d'avoir un système simulé, on remplace le système par un autre système (qui est "émulé"). Cette distinction rejoint celle faite au travers de la notion de modèle d'émulation (cf Définition 2.2). De plus, cette définition semble cohérente avec sa signification en électronique. Par exemple, au lieu que le code soit exécuté sur un micro-processeur réel, il est exécuté sur un ordinateur via un logiciel qui reproduit le comportement du micro-processeur. La plateforme matérielle d'exécution réelle du code est donc remplacée par une autre plateforme d'exécution qui reproduit les interactions réelles. On retrouve cette signification aussi en informatique où l'on peut utiliser un émulateur d'un système d'exploitation, par exemple. L'émulateur est une machine virtuelle de la plateforme cible qui permet d'exécuter des programmes de la plateforme cible. Ainsi, dans le contexte de l'électronique ou de l'informatique, l'émulation consiste à remplacer une plateforme "matérielle" d'exécution par autre chose qui reproduit son fonctionnement, somme toute une autre plateforme d'exécution.

De notre point de vue, puisque dans le cadre de l'émulation on remplace un système (une plateforme d'exécution) par un autre système, l'approche dite d'*émulation* en génie automatique correspond à une approche de test de type Software-in-the-loop (Figure 2.2) : le programme de commande réél est exécuté sur une plateforme d'exécution émulée, le système à commander est quant à lui simulé. Nous considérons ainsi que vis-à-vis des tests à effectuer, il ne s'agit pas uniquement de reproduire la réponse du système à commander (réel) mais également d'observer son état interne [1].

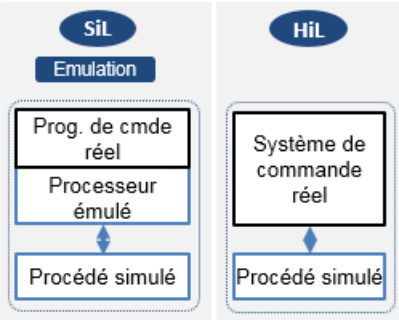


FIGURE 2.2 – Approche de test par *émulation*

2.1.2.2 Virtual commissioning

Dans le cadre des systèmes de production, afin de limiter la détection tardive d'erreurs dans les systèmes de contrôle-commande, notamment pendant la phase de tests en situation réelle (*commissioning*), il a été introduit une phase de tests "virtuels" (*virtual commissioning*). Dans cette étape de tests virtuels, on retrouve les principes de l'approche d'*émulation* tel qu'évoqués par le génie automatique. Il s'agit de tester le système de contrôle-commande sur un système de production *virtuel*. L'intérêt du *virtual commissionning*, est de séparer la partie simulation, du système de commande à vérifier. En fonction de l'activité de V&V par

1. S'il s'agit uniquement de reproduire la réponse du système à commander, le système à commander est alors émulé. L'approche d'émulation peut dans ce cas correspondre à une approche de type Hardware-in-the-loop si le processeur réel est utilisé.

rapport au cycle de conception et du domaine, diverses classifications des types de plate-forme de simulation sont apparues, selon les configurations de partie simulée et partie réelle. Ainsi selon le cas, une approche Software-in-the-loop peut être utilisée ou Hardware-in-the-loop [Puntel-Schmidt and Fay, 2015]. Par exemple, pour [Lee and Park, 2014, Fratczak et al., 2015, Süß et al., 2015], c'est le système de contrôle-commande réel entier qui est utilisé, incluant ainsi l'automate programmable industriel cible. Au contraire, pour [Kim et al., 2013], l'automate programmable industriel est émulé. Quoi qu'il en soit, le *virtual commissioning* apparaît comme une phase supplémentaire de tests intégrée dans le processus de conception d'un système complexe.

L'idée de [Oppelt and Urbas, 2014] est d'introduire ces tests, non pas comme une phase précédant les tests réels, mais pendant la conception du système, c'est-à-dire d'intégrer le *virtual commissioning* au processus de conception (*integrated virtual commissioning*). Cela passe entre autre, par l'utilisation de modèles relativement simples au début, qui sont enrichis au fur et à mesure que l'on avance dans la conception. Le *virtual commissioning*, tel qu'il est décrit dans [Oppelt and Urbas, 2014], n'est donc plus une étape supplémentaire dans le processus de conception, mais une tâche qui s'effectue en parallèle du travail de l'automaticien, afin qu'il puisse vérifier et valider son travail. Cette idée était aussi évoquée par [Drath et al., 2008] où les exigences pour une utilisation dans un contexte industriel sont définies. L'intérêt d'intégrer le *virtual commissioning* au sein des activités de conception, comme décrit par [Dahl et al., 2016], est de pouvoir permettre le test du système de commande avant la construction du système physique, mais aussi de pouvoir effectuer les tests de façon continue.

2.1.3 Bilan

En se plaçant dans un contexte de conception simultanée, chacune des trois approches de test par simulation identifiées (à savoir Model-in-the-loop, Software-in-the-loop et Hardware-in-the-loop) peuvent être mises en oeuvre à différents endroits du cycle de conception (Figure 2.3).

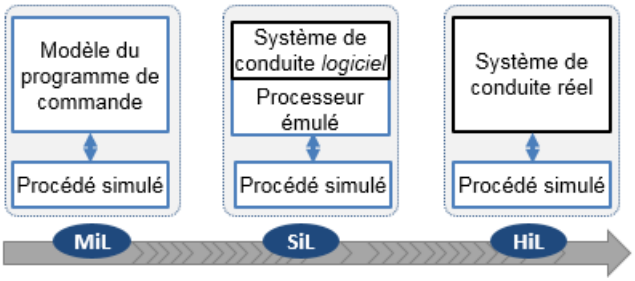


FIGURE 2.3 – Synthèse des approches tests d'un système de conduite

Une approche de type MiL peut être mise en oeuvre en début de conception, dès qu'un modèle de la commande est disponible afin de vérifier la logique de commande et la valider. On s'intéresse donc au principe de solution. Puis, une fois la partie logicielle implémentée, on peut mettre en oeuvre une approche SiL, afin de vérifier et valider le système de contrôle-commande.

Dans ce type d'approche, on peut, en outre, vérifier le système de conduite dans son ensemble (l'automate programmable industriel étant émulé). On peut donc effectuer des vérifications au niveau du contrôle-commande, au niveau de l'IHM (notamment par rapport aux interactions Homme-Machine), mais on peut également vérifier le système Homme-Machine comme un tout. Enfin la dernière approche, HiL, s'effectue plutôt en fin de conception, notamment dans le cadre d'une phase de tests virtuels avant de réaliser des tests réels. Dans cette dernière approche, le procédé virtuel peut être réalisé à un niveau de détail très précis (le système physique étant figé).

Ainsi, ces trois approches de test peuvent être utilisées lors de la mise en oeuvre de tests en parallèle du travail des concepteurs (dans le cadre du *integrated virtual commissioning*). Cependant, il peut potentiellement y avoir un fossé entre une approche MiL et SiL par rapport aux besoins de modélisation (en terme d'abstraction du procédé, et des techniques de simulation).

Dans un contexte de conception participative et d'ingénierie des systèmes sociotechniques, l'utilisation d'une approche de type SiL semble adaptée au besoin de vérification et validation du système de conduite de procédé, au plus tôt, en articulant une vision technocentrée et une vision anthropocentrée, lors de situations dynamiques. Toutefois, avant de pouvoir utiliser la simulation pour effectuer les tests, il faut disposer du "système de simulation". Il faut donc concevoir et construire ce système par rapport à l'usage que l'on souhaite en faire. La section suivante s'intéressera ainsi à l'obtention de la simulation.

2.2 La simulation : une activité

Comme nous l'avons évoqué, il existe de nombreuses et diverses définitions de la *simulation* ainsi que de nombreux types d'application. Dans le cadre de nos travaux, on s'intéresse à la simulation informatique, qui consiste à reproduire ou représenter une partie (ou un point de vue) du monde réel (ou d'un système) par un programme exécutable. Pour ce faire, elle s'appuie sur un modèle, que l'on fait évoluer à travers le temps, afin de reproduire un comportement.

D'après [Varenne, 2010] la simulation peut être caractérisée comme « *une stratégie de symbolisation prenant la forme d'au moins un traitement étape par étape. Ce traitement se décompose en deux phases majeures : une phase opérative et une phase d'observation. La simulation informatique est une simulation pour laquelle la phase opérative est au moins réalisée sur un calculateur numérique et programmable* ». La simulation peut ainsi être vue comme l'exécution d'un modèle à travers le temps, à l'aide de règles d'évolution. Il faut donc disposer d'un modèle exécutable (de ce que l'on souhaite simuler) pour pouvoir réaliser une simulation. Pour développer et implémenter ce modèle exécutable, une phase de modélisation est nécessaire. La simulation est donc étroitement liée à la notion de modélisation. La conception d'une simulation est ainsi une activité interdisciplinaire, il faut d'une part avoir des connaissances en modélisation, sur le système que l'on souhaite modéliser (qui fait potentiellement intervenir différents domaines métiers) et en informatique (implémentation et

exécution du modèle).

Comme le souligne Page, la manière dont un modèle de simulation est conçu, développé et utilisé, a un impact significatif sur la capacité du modèle à remplir ses objectifs [Page et al., 2000]. Ainsi, à l'image de la conception d'un système, la simulation suit elle aussi un processus de conception et développement.

2.2.1 Processus de conception d'une simulation

De façon assez générale, le processus de conception d'une simulation est constitué de trois phases principales : une première phase de modélisation, puis une seconde d'implémentation qui permet d'obtenir le modèle exécutable, et enfin une phase de tests (appelée généralement phase d'expérimentations) qui permet de tester le système de simulation [2]. Les processus de modélisation et de simulation selon [Frantz, 1995] et [Sargent, 2011] en sont des illustrations. Le système du monde réel (*Système d'intérêt* sur la Figure 2.4) qu'il soit existant ou hypothétique, correspond au système que l'on souhaite simuler pour conduire des expérimentations. Il s'agit ainsi du système que l'on souhaite modéliser pour pouvoir répondre à certaines questions. A partir de ce système du monde réel, la première étape, de modélisation, permet d'obtenir le modèle conceptuel, tandis que la deuxième étape, d'implémentation, permet d'obtenir le modèle de simulation (modèle exécutable).

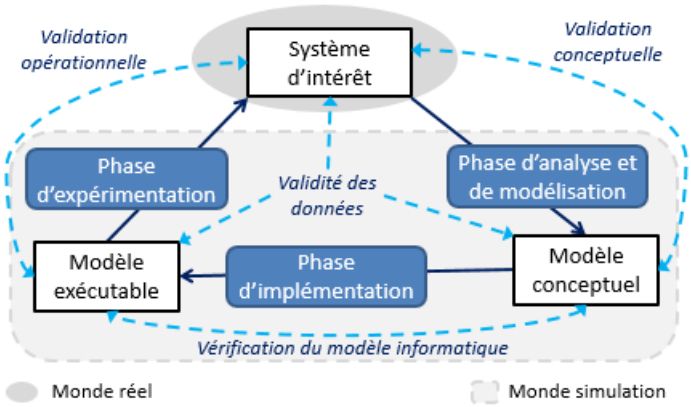


FIGURE 2.4 – Processus de modélisation et de simulation simplifié selon [Sargent, 2011]

Le **passage du monde réel vers un modèle conceptuel** se fait via une phase d'analyse et de modélisation. Avec un mécanisme d'*abstraction* [3] sur le système et en faisant des *hypothèses* sur les données observables [4], on entre dans le domaine du *monde de la simulation*. En associant ces derniers avec l'*objectif d'utilisation* de la simulation, un modèle conceptuel est construit. Le *modèle conceptuel* est, selon Sargent, une représentation mathématique/logique/-verbale du système d'intérêt pour une étude particulière [Sargent, 2011]. Selon Fishwick, un

2. Le *système de simulation* comprend le modèle exécutable et l'environnement qui permet de l'exécuter.
3. Le mécanisme d'abstraction permet de réduire la complexité du modèle sans changer la validité des résultats de la simulation [Frantz, 1995].
4. Les sources de données observables sont obtenues en réalisant des expérimentations sur le système réel [Sargent, 2001].

modèle conceptuel correspond à la première phase dans les efforts de modélisation. On cherche à y retranscrire la connaissance statique et dynamique du système (physique), sous une forme permettant la spécification des interactions sans avoir à spécifier quantitativement la dynamique [Fishwick, 1995]. Dans cette phase de modélisation, comme le rappelle Frantz, il s'agit de déterminer, entre autres, quels sont les facteurs influençant le comportement du système, les comportements qui doivent être modélisés et leurs représentations. Cette détermination dépend de l'objectif d'utilisation de la simulation ; des ressources disponibles (notamment en termes de puissance de calcul) pour la construction, la validation et l'utilisation ; ainsi que des données disponibles pour décrire le système du monde réel et ses interfaces [Frantz, 1995]. Puis une *validation conceptuelle* permet de s'assurer que les hypothèses de modélisation sont correctes par rapport à l'objectif d'utilisation et que c'est le "bon" modèle conceptuel qui a été construit.

Le **passage du modèle conceptuel vers un modèle exécutable** se fait via des étapes successives de spécification et d'implémentation. Il est à noter que le terme *modèle* (de simulation) est traditionnellement utilisé dans la communauté de *Modélisation et Simulation* pour faire référence à un modèle informatique (*computational model*) sous la forme d'un programme (code) et non en référence à un modèle exprimé dans un langage de modélisation graphique de haut niveau, comme c'est le cas dans le domaine des *Systèmes d'information* et du *Génie Logiciel* [Wagner et al., 2016]. Le *modèle de simulation* y est ainsi vu comme un programme (du code) alors que le *modèle conceptuel* est plutôt perçu comme un modèle mental ou schématique. Pour s'assurer que le *modèle de simulation* est correct par rapport au *modèle conceptuel*, une *vérification du modèle informatique* est effectuée. Des mécanismes de vérification permettent de s'assurer que le modèle de simulation est une représentation fidèle du modèle conceptuel [Frantz, 1995].

Toutefois, selon Cardin, la vérification revient à répondre à la question « est-ce que le modèle de simulation fonctionne correctement ? » [Cardin, 2007]. La vérification ne consiste pas uniquement à garantir que le passage du modèle conceptuel vers le modèle exécutable est correct. Il s'agit également de réaliser des tests de bon fonctionnement, que ce soit au niveau du modèle exécutable ou du système de simulation dans son ensemble [5].

Ces tests de bon fonctionnement correspondent à la **phase d'expérimentation** dans le processus de conception tel que défini par Sargent (Figure 2.4). Cette phase s'intéresse au système de simulation dans son ensemble (le modèle exécutable et ce qui permet de l'exécuter). Pour Sargent, les expérimentations permettent de déterminer si les résultats de la simulation sont adaptés et assez précis pour l'objectif d'utilisation (*validation opérationnelle*). D'autre part, il faut pouvoir garantir la *validité des données* entre le système d'intérêt et les modèles, que ce soit pour construire le modèle, l'évaluer, le tester ou lors de son utilisation (par rapport à l'objectif donné). Frantz parle, quant à lui, de *crédibilité* pour garantir la qualité liée à l'usage de la simulation (lors de l'exécution de la simulation). Il s'agit de se demander

5. Il s'agit, en particulier de vérifier le bon fonctionnement du modèle de simulation et de s'assurer que les règles implantées dans le logiciel de simulation décrivent les flux comme spécifiés dans la phase de modélisation, [Cardin, 2007].

« Quelle confiance l'utilisateur peut-il avoir dans les résultats de la simulation ? ». Ceci passe, entre autre, par avoir confiance dans le système de simulation, recouvrant ainsi les notions de *validation opérationnelle* et de *validité des données*. De façon plus générale, pour avoir confiance dans le modèle de simulation, les modèles (conceptuel et exécutable) doivent être correctement construits (notion de vérification), et ce sont les *bons* modèles qui doivent avoir été construits (notion de validation).

Le développement d'une simulation suit un processus de développement itératif (modélisation, implémentation et expérimentation), intégrant des processus de vérification et validation. Tandis que les processus de conception de Frantz et Sargent mettent l'accent sur l'obtention du modèle de simulation, celui de Zeigler sépare le modèle de ce qui permet de l'exécuter [Zeigler et al., 2000]. En particulier, au travers de la notion de *simulateur* (Figure 2.5). Un *simulateur* y est défini comme un système de calcul obéissant à des instructions pour exécuter un modèle et en produire son comportement. La *relation de simulation* permet de vérifier qu'en obéissant aux instructions définies dans le modèle de simulation, le simulateur reproduit le comportement spécifié. Le *cadre expérimental* spécifie les conditions d'observations du système et les objectifs de la simulation. Le système source (le système que l'on veut modéliser) peut être réel ou virtuel, il constitue la source de données observables. Ces données observables forment la base de données comportementales. La *relation de modélisation* permet ainsi d'établir la validité de l'expérience [Foures, 2015].

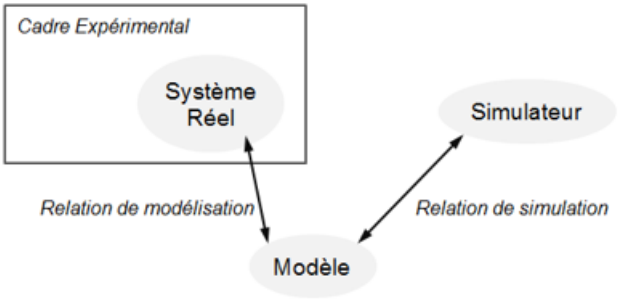


FIGURE 2.5 – Processus de M&S selon B.P. Zeigler

Cette représentation du processus de Modélisation et de Simulation (2.5) apporte ainsi un complément aux processus de conception d'une simulation, en distinguant clairement le modèle de simulation et l'environnement d'exécution. De façon générale, pour obtenir une simulation, on distingue une phase de modélisation (au sens large) qui permet d'obtenir le *modèle de simulation*, de la phase de simulation qui concerne l'exécution du modèle. Cette phase de modélisation comprend ainsi le passage du *monde réel* vers le *modèle conceptuel* et le passage du *modèle conceptuel* vers le *modèle de simulation*.

2.2.2 Evolutions des approches et outils pour la simulation

Depuis les débuts de la simulation sur ordinateur, la communauté scientifique s'est intéressée aux méthodologies de modélisation pour la simulation [Page et al., 2000]. Contrairement aux pionniers de la simulation qui n'avaient pas d'autre choix que de tout programmer pour pouvoir mener des simulations, avec l'apparition des "simulation worldviews" (les visions du

monde pour la simulation), le développement du modèle de simulation est séparé de celui du moteur de simulation [Pidd, 2004]. Les pratiques liées à la construction des modèles de simulation ont ainsi subi de grands changements.

Comme le fait remarquer Page, entre les années 1960 et 2000, ces pratiques sont passées :
- de la programmation (codage) dans des langages à usage généraux (FORTRAN, C...) ;
- au développement de modèles dans des langages de simulation à usage spécifique ;
- à la conception de modèles en utilisant des langages de spécification de modèle de simulation et formalismes de plus haut niveau ;
- à des théories générales de modélisation pour la simulation et le support d'environnement holistique pour la tâche de modélisation.

On constate ainsi un basculement d'activités principalement de programmation vers des activités de modélisation pour la simulation.

2.2.2.1 Un problème de modélisation avant tout

Avec les évolutions des techniques et technologies, des solutions logicielles permettent de faciliter l'implémentation. L'obtention du modèle de simulation est ainsi avant tout un problème de modélisation, et de choix technologiques (quel langage de simulation et quel environnement d'exécution utiliser) par rapport au problème à résoudre (au besoin). Bien que la notion de *modèle conceptuel* soit relativement ambiguë dans la littérature, tous s'accordent à dire que c'est la première étape pour obtenir le modèle de simulation [Robinson, 2004]. En effet, comme précédemment évoqué, le *modèle de simulation* est une implémentation du *modèle conceptuel*.

Cette étape de modélisation, est ainsi l'une des plus importantes dans le projet de simulation [Robinson, 2008a]. C'est notamment là que sont déterminés les objectifs de la modélisation, quelles seront les entrées/sorties de la simulation et le contenu du modèle de simulation (ce qui doit être modélisé), et que l'on récupère les données nécessaires pour développer le modèle de simulation. Cependant, Robinson constate que cette étape de modélisation est plutôt perçue comme un art plutôt qu'une science et que, en tout état de cause, l'art de la modélisation (*conceptual modelling*) est acquis par l'experience [Robinson, 2004]. Une définition plus précise de ce qu'est un *modèle conceptuel* a ainsi été donnée.

Définition 2.3 (modèle conceptuel [Robinson, 2004])
Le *modèle conceptuel* est une description du modèle de simulation (qui doit être développé), indépendante de toute plateforme logicielle, qui décrit : les objectifs de modélisation, les entrées/sorties, le contenu du modèle (scope du modèle et niveau de détail), ainsi que les hypothèses de modélisation (liées à la connaissance du système et/ou aux incertitudes) et les simplifications (qui sont faites afin de créer des modèles plus simples).

Cette définition a été enrichie par la suite, en considérant que le *modèle conceptuel* est une description du *modèle de simulation* qu'il soit celui à développer, qui est développé ou bien qui a été développé [Robinson, 2008a]. Cette définition semble cohérente, avec celle utilisée

dans le cadre de la simulation de flux pour les systèmes de production [6].

Puis, afin de guider l'élaboration du modèle conceptuel, Robinson a proposé un cadre méthodologique (*framework for conceptual modelling*) [Robinson, 2008b]. Cinq activités clés y sont définies :

- comprendre la situation problématique,
- déterminer les objectifs du projet et de la modélisation,
- identifier les sorties du modèle,
- identifier les entrées du modèle,
- déterminer le contenu du modèle (scope et niveau de détail), identifier les hypothèses de modélisation et simplifications

Bien évidement, l'ordre de réalisation de ces activités n'est pas strict, plusieurs itérations sont souvent nécessaires. Les deux premières activités visent à définir les objectifs de la simulation : pourquoi a-t-on besoin de la simulation (comprendre la situation problématique), qu'est-ce que l'on attend de la simulation (objectifs de modélisation) et quels sont les objectifs d'utilisation (objectifs généraux du projet). Après avoir défini les objectifs, il s'agit ensuite d'identifier quelles sont les entrées/sorties du modèle (approche de type boîte noire). Puis, de déterminer le contenu du modèle (approche de type boîte blanche), c'est-à-dire que doit-on modéliser et comment.

Cependant, déterminer le contenu du modèle demande des compétences en modélisation. Il s'agit notamment de comprendre la structure et les règles de fonctionnement du système d'intérêt, et être capable d'extraire l'essence du système sans inclure de détails inutiles [Pritsker et al., 1991]. En effet, plus le niveau de détail est élevé, plus le niveau d'abstraction du modèle est faible, et par conséquence plus le modèle contiendra une grande quantité d'information [Benjamin et al., 1998]. Or, il n'est pas forcément pertinent de tout représenter selon le but poursuivi. Le choix du niveau de détail (niveau d'abstraction) détermine la quantité d'information mais pas la pertinence des informations [Benjamin et al., 1998]. Le mécanisme déterminant quelles informations sont pertinentes par rapport à un objectif de modélisation, qui est appelé perspective dans [Benjamin et al., 1998], semble ainsi être similaire à ce que Robinson appelle le scope. Toutefois, le mécanisme d'abstraction est fortement lié au mécanisme déterminant la pertinence des informations. Ainsi afin d'obtenir le modèle le plus simple possible, différents niveaux d'abstraction sont potentiellement nécessaires. Une grande difficulté dans l'obtention du modèle réside donc dans le choix de ce qu'il faut représenter : quels éléments du système doivent être représentés (le scope) et à quel niveau de détail [Robinson, 2012].

Une fois le modèle conceptuel obtenu, on peut passer à la construction du modèle de simulation à proprement parler. Pour cela, il existe différents environnements logiciels qui permettent de faciliter le développement.

6. En effet, selon [Fontanili et al., 2008], le *modèle conceptuel*, qui permet de capitaliser la connaissance indépendamment du modèle informatique, doit répondre à des objectifs multiples, comme : structurer et rassembler les données utiles à la construction du modèle de simulation, représenter des flux physiques et informationnels, identifier et définir les règles et lois utiles au pilotage du système, servir de support pour la réalisation du modèle de simulation, servir d'outil de communication entre les différents acteurs.

2.2.2.2 Les outils pour la simulation

Pour faciliter la conception d'une simulation, et en particulier la construction du modèle de simulation, des solutions logicielles ont été proposées, permettant de s'affranchir en grande partie du langage de programmation, que ce soit dans la phase de modélisation ou dans la phase d'expérimentation. C'est notamment grâce à l'apparition de la simulation interactive et visuelle, *Visual Interactive Simulation* (VIS), introduite dans les années 1970, qu'il a été permis de s'affranchir en grande partie du langage de programmation [Fontanili et al., 2008]. Mais aussi grâce à la modélisation interactive et visuelle, *Visual Interactive Modeling* (VIM) qui vise à faciliter le développement du modèle de simulation via l'utilisation d'un langage graphique, le modèle graphique obtenu est ensuite traduit en code exécutable [Wagner et al., 1996]. Dans ce type d'approche, on construit donc un modèle graphique intermédiaire.

Contrairement aux idées reçues (du moins au début des années 1990), la simulation interactive et visuelle (VIS) n'est pas forcément une façon de construire les modèles de simulation, mais est principalement une méthodologie de résolution de problème via l'utilisation de la simulation [Bell and O'Keefe, 1994]. Pour cela, des représentations graphiques et animations sont introduites pour permettre à l'utilisateur de visualiser le comportement durant l'exécution de la simulation. Un des concepts clés concerne le *modèle simulation* : ce n'est plus uniquement du code, il comporte également une représentation graphique. Il y a donc un *modèle symbolique* (le code) et un *modèle visuel* (une représentation graphique) qui lui est associé.

On retrouve deux approches dans ce type de méthodologie, l'une où la représentation graphique modélise tous les éléments du modèle symbolique, l'autre où seulement les portions "utiles" (pour l'utilisateur) du modèle symbolique sont représentées graphiquement [Bell and O'Keefe, 1994]. Dans une méthodologie VIS, l'utilisateur de la simulation prend ainsi part au processus de modélisation (d'implémentation), même si ce n'est pas lui qui construit le modèle de simulation. Toutefois, ce type d'approche se focalisant plutôt sur l'utilisation de la simulation que sur l'implémentation du modèle de simulation, d'autres approches sont apparues pour que l'utilisateur de la simulation puisse lui-même construire le modèle de simulation.

Par exemple, des packages de simulation, souvent appelés "simulateurs", permettent à l'utilisateur de construire et exécuter des modèles de simulation (de type VIS), pour des classes limitées de problèmes [Bell and O'Keefe, 1994]. A noter que ce type de logiciel est appelé *logiciel prêt à l'emploi* par [Pidd, 2004]. Dans cette méthodologie [7], l'utilisateur ne conçoit pas le modèle symbolique, car il est déjà présent dans le logiciel, mais il construit le modèle graphique. Une autre méthodologie [8] apporte une aide pour construire le modèle de simulation (de type VIS) en utilisant un langage graphique. Deux modèles graphiques sont ainsi utilisés, un modèle graphique qui permet d'obtenir le modèle symbolique, le second modèle graphique qui concerne la visualisation du comportement du modèle symbolique pendant l'exécution de

7. Qualifiée de *Interactive Simulation Modeling* par [Bell and O'Keefe, 1994].
8. Qualifiée de *Graphical Simulation Modeling* par [Bell and O'Keefe, 1994].

la simulation. De façon générale, l'introduction de graphiques et d'interactions dans toutes les étapes du processus de modélisation et de simulation est appelée modélisation et simulation interactive et visuelle, *Visual Interactive Modeling and Simulation* (VIMS) [Čerić, 1997]. Elle est ainsi basée sur les méthodologies VIS et VIM (utilisation d'une représentation graphique pour obtenir le modèle de simulation, et visualisation des résultats pendant l'exécution et après). L'intérêt est de permettre à des non-experts en programmation de pouvoir développer rapidement une simulation et de pouvoir l'utiliser pour conduire des expérimentations [Pidd, 2004]. Nombre de logiciels de simulation actuels sont basés sur cette méthodologie. Il est à noter que ce type de logiciel est basé sur l'utilisation d'un langage de simulation, même lorsqu'il s'agit de logiciel prêt à l'emploi [Pidd, 2004].

On retrouve ainsi des environnements logiciels pour la simulation basés sur des méthodologies VIS, VIM, ou VIMS. Ainsi, pour implémenter un modèle de simulation, il existe de façon générale trois possibilités. On peut par exemple utiliser la programmation, comme cela était le cas dans les années 1950, utiliser un langage de simulation ou bien un logiciel de simulation "prêt à l'emploi". L'avantage de la *programmation* est la flexibilité, par contre sa mise en oeuvre sera fastidieuse, d'autant plus qu'il faudra aussi potentiellement développer le moteur de simulation (le simulateur). L'utilisation d'un *langage de simulation* permet, entre autres, d'utiliser des primitives déjà existantes, ce qui laisse une certaine flexibilité, par contre les coûts de développement restent importants. Enfin, en utilisant un *logiciel de simulation prêt à l'emploi*, on aura peu de programmation car le comportement est pré-programmé (puisqu'un modèle de simulation générique est fourni). Néanmoins, les usages sont limités, ce qui implique de trouver le logiciel adapté.

Ainsi les logiciels de simulation actuels fournissent non seulement un environnement pour simuler, mais aussi un environnement de modélisation. L'implémentation du modèle en programme exécutable se fait ainsi de façon relativement transparente pour le concepteur/utilisateur. L'obtention de la simulation n'est plus une activité essentiellement de programmation, mais surtout un problème de modélisation. Les logiciels de simulation modernes permettent de rapidement développer le code du modèle de simulation, mais ils ne permettent cependant pas de s'afranchir de la conception du *modèle conceptuel* [Robinson, 2008a]. Une des plus importantes difficultés pour obtenir le modèle de simulation est ainsi de déterminer ce qu'il doit contenir (c'est-à-dire ce qui doit être modélisé) [Robinson, 2015].

Toutefois, il peut y avoir un écart important, en terme de représentation notamment, entre le modèle conceptuel et le modèle de simulation (en tant que représentation graphique) que l'on saisit dans l'environnement de modélisation pour la simulation.

2.2.3 Modélisation pour la simulation

Le modèle conceptuel étant indépendant du modèle de simulation, le contenu du modèle doit ainsi être défini sans tenir compte de la plateforme cible d'exécution du modèle de simulation ([Fishwick, 1998], [Robinson, 2004],[Fontanili et al., 2008], [Roca et al., 2015]). Ainsi, l'approche de modélisation pour le modèle conceptuel peut être très différente de celle retenue

pour le modèle de simulation. Le modèle conceptuel peut ainsi prendre différentes formes (en termes de représentation), selon la méthode de modélisation utilisée. Prenons l'exemple d'une simple file d'attente donnée par [Robinson, 2004] où quatre méthodes de modélisation sont distinguées, à savoir : via une liste de composants, via un diagramme de flux (Process Flow Diagram), via un diagramme logique (Logic Flow Diagram), via un diagramme d'activité (Activity Cycle Diagram). Ainsi selon le choix retenu et les choix technologiques, une étape de modélisation intermédiaire peut être nécessaire pour passer du modèle conceptuel vers un modèle de simulation, notamment au travers d'un modèle pour la conception de la simulation.

Prenons par exemple, la simulation de flux [9] qui est largement répandue dans le domaine des systèmes de production. Deux types d'approches de modélisation sont utilisés pour obtenir le modèle de simulation : une approche composant (représentation du comportement des composants du système), et une approche orientée processus (représentation fonctionnelle, de plus haut niveau). Dans une approche *composant*, la modélisation se fait à partir d'un ensemble de composants de base (entité). Le comportement intrinsèque du composant est décrit à l'aide d'attributs statiques et dynamiques. Cela permet notamment de faire coïncider les éléments du modèle avec ceux du système réel. Selon [Revel et al., 2004], les avantages s'ajoutant à ceux d'une modélisation orientée objet (mécanisme d'héritage, modularité, réutilisabilité ...) pour les outils basés sur ce type de modélisation sont :

- approche naturelle et facile à comprendre,
- possibilité d'inclure une très large variété d'éléments d'un système, notamment concernant le caractère comportemental de celui-ci,
- possibilité d'appréhender le système selon différents niveaux d'abstraction,
- combinaison des avantages des langages généraux (flexibilité et applications larges) aux avantages d'outils spécialisés (objets paramétrables, et construction simple des modèles),
- absence de compilation, ce qui permet des performances élevées de l'outil de simulation.

Cependant, ce type d'outil reste principalement destiné à des spécialistes de la simulation. Contrairement à l'approche *composant*, dans une approche *orientée processus* (approche fonctionnelle), on utilise une bibliothèque de fonctions, aussi appelées *primitives*. Ces primitives peuvent être tant généralistes que spécialisées. La modélisation dans ce cas est orientée processus, c'est-à-dire que l'on ne s'intéresse pas aux entités du système en tant que telles, mais à leur fonction. D'après [Muzy and Hill, 2011], un *processus* est une séquence d'*activités* ou d'*événements* ordonnés dans le temps. Une *activité* est une opération qui transforme l'état du système au cours du temps. Elle commence par un *événement* et se termine en produisant un *événement* (de fin d'activité), un *événement* étant ce qui provoque un changement de l'état du système (interne ou externe). La modélisation dépend ainsi fortement des primitives qui sont proposées, et du besoin. Lorsque les fonctions proposées sont abstraites et génériques, la modélisation est complexe (et par conséquent réservée aux spécialistes) mais permet d'avoir un large domaine de simulation [Revel et al., 2004]. A contrario, lorsque les fonctions pro-

9. La simuation de flux est généralement utilisée pour dimensionner les systèmes de production, le choix des politiques de gestion de production, mais aussi la validation des parties logicielles [Revel et al., 2004].

posées sont spécialisées, la modélisation est plus simple (potentiellement non réservée aux spécialistes) mais le domaine de simulation devient restreint. Ainsi, l'approche de modélisation pour obtenir le modèle de simulation semble plutôt liée à des caractéristiques d'exécution de la simulation.

Malgré un désir de clairement séparer la modélisation de la simulation (exécution du programme de simulation), la distinction entre méthode d'exécution et méthode de modélisation ne semble pas toujours évidente, au regard de la littérature. Cela se ressent au travers de la notion de *modèle* (de simulation) qui parfois fait référence à du code et parfois à une représentation graphique de plus haut niveau (et voire même aux deux), mais aussi au niveau des taxonomies traditionnelles des modèles de simulation qui sont souvent liées au paradigme d'implémentation de la simulation plutôt qu'à un paradigme de modélisation.

Traditionnellement, on distingue trois catégories de simulation [10], en fonction de la façon dont évoluent les variables d'états du système : statique, continue, à événements discrets.

- La simulation *statique* correspond à des variables d'états n'évoluant pas en fonction du temps, et est plus connue sous le nom de simulation de Monte-Carlo. La modélisation des phénomènes aléatoires se fait en utilisant des variables aléatoires et des lois de probabilités.
- La simulation *continue* correspond à des variables d'états qui évoluent de façon continue en fonction du temps. La modélisation se fait sous forme de systèmes d'équations, où le temps et les relations d'états ne sont pas précisément représentés [Nance, 1993].
- Et pour terminer, la simulation discrète qui est aussi appelée simulation *à événements discrets*, correspond à une évolution des variables d'états lors d'événements discrets. Elle peut être dirigée par les événements (*event-driven*) ou par le temps (*time-driven*) [Ray, 2003]. La modélisation du système physique se fait avec des modèles mathématiques et/ou logiques qui représentent les changements d'états en des instants précis du temps simulé, ce qui implique une description précise de la nature du changement d'état et de son occurrence [Nance, 1993].

Dans le cas de la simulation continue, des techniques d'intégration sont utilisées pour obtenir l'évolution continue des variables d'état. Dans le cas de la simulation à événements discrets dirigée par le temps, l'horloge avance par intervalle fixe. A chaque pas d'horloge, la liste d'événements est consultée, et ceux qui doivent l'être sont exécutés. A contrario, quand la simulation est dirigée par les événements, le temps est incrémenté en fonction de l'occurrence des événements (le pas d'horloge n'est pas fixe).

Généralement, c'est à partir de cette taxonomie traditionnelle de simulation, que la taxonomie des modèles de simulation est définie [Miller et al., 2004]. Cependant, comme le souligne Fishwick, ces taxonomies traditionnelles reflètent la méthode d'exécution du modèle plutôt que la structure de conception du modèle [Fishwick, 1998]. Il a donc proposé de séparer clairement la conception du modèle (qui pour lui représente la syntaxe) de son exécution (qui représente la sémantique). D'autre part, une distinction entre un *modèle de simulation* et un

10. Il existe également d'autres formes de simulation. Il s'agit d'approches mixtes résultant de combinaisons entre les différentes approches issues de cette classification.

programme de simulation (ou code) est faite par Fischwick. En effet, il considère un *modèle* comme une représentation graphique de haut niveau d'un système physique, tandis que les constructions de bas niveau définissant la dynamique du système sont des programmes ou des spécifications formelles [Fishwick, 1998]. Bien qu'il distingue un modèle d'un programme, il reconnaît l'existence de liens entre modélisation et programmation : ils utilisent tous les deux des langages. Toutefois, il souhaite ainsi mettre en évidence les similitudes entre *langage de modélisation* et *langage de programmation*. En effet, modélisation et programmation reposent sur la notion de *langage*. Un *langage* est par définition constitué d'une part par sa syntaxe qui décrit les constructions du langage ainsi que les règles d'agencement de celles-ci, et d'autre part, par sa sémantique qui permet de donner un sens à chacune des constructions du langage [Combemale, 2008].

Fishwick propose ainsi une taxonomie pour la simulation basée non pas sur des caractéristiques d'exécution du modèle mais sur le modèle en lui-même (sa forme), permettant ainsi de faire le lien entre les paradigmes de programmation et de modélisation [Miller et al., 2004]. Pour illustrer son propos, Fishwick nous donne quelques exemples [Fishwick, 1998]. Ainsi, un modèle de simulation *déclaratif* est un modèle où sont directement spécifiés les états et les transitions d'événements (automates, Réseau de Petri ...), tandis qu'une sémantique *déclarative* de programmation spécifie l'évolution des valeurs des variables par déclaration des valeurs présentes et futures. Un modèle *fonctionnel* représente des flux dirigés alors qu'une sémantique *fonctionnelle* spécifie l'évolution des variables par des fonctions (au sens mathématique du terme).

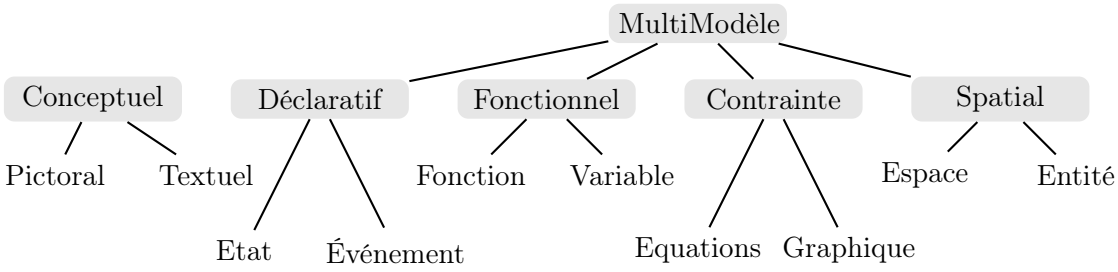


FIGURE 2.6 – Taxonomie des modèles pour la simulation selon Fishwick [Fishwick, 1998]

La Figure 2.6 illustre sa proposition de taxonomie pour la modélisation. Bien que ce ne soit pas un modèle exécutable, cette taxonomie fait apparaître le modèle Conceptuel, qui peut être de forme picturale ou en langage naturel. Il représente les hiérarchies d'aggrégation et de généralisation (comme dans une approche orientée objet). Ce modèle est considéré comme la première étape pour obtenir un modèle qui contient d'autres modèles, pouvant être de nature différente. Le modèle Déclaratif s'intéresse aux motifs (*patterns*) associés aux états (ex : automate à états finis) ou aux événements (ex : graphe d'événements). Le modèle Fonctionnel est un réseau dans lequel les composants principaux sont soit des fonctions (fonctions de transferts sous forme de blocs ou de boîtes), soit des variables. Le modèle de Contraintes est soit sous forme d'un ensemble d'équations (équations différentielles ordinaires, équations aux différences ...), soit sous forme graphique (comme par exemple les Bond Graphs). Enfin, le

modèle Spatial se focalise soit sur l'espace dans son ensemble (*whole space*) en utilisant les équations aux dérivées partielles par exemple, soit sur des entités.

Au travers de cette taxonomie, il ressort une notion importante, celle de *multi-modèle*, à savoir un modèle contenant des sous-modèles de nature potentiellement différente. Les travaux concernant la modélisation pour la simulation combinée sont à l'origine de ce concept de *multi-modèle*, qui a été introduit dans les travaux de [Ören, 1991]. Il s'agissait d'un modèle contenant des sous-modèles dont seulement un était actif à un instant donné. Actuellement, on considère que selon le type de *multi-modèle*, seulement un sous-modèle est actif, ou bien que tous les sous-modèles ou plusieurs d'entre eux peuvent être actifs en même temps [Ören, 2014]. En effet, pour reproduire le comportement d'un système, il est souvent nécessaire d'utiliser différents paradigmes de modélisation pour capturer les différentes facettes du système d'intérêt. Ainsi, cette taxonomie permet, entre autres, d'introduire la notion de modélisation hétérogène, sans toutefois se préoccuper de comment exécuter le modèle de simulation (que ce soit via la co-simulation où plusieurs simulateurs sont couplés, ou via l'utilisation d'un seul simulateur supportant la multi-modélisation).

Un lien entre développement d'une simulation et l'Ingénierie Dirigée par les Modèles se dessine également. Celle-ci pourrait être utilisée pour faciliter l'obtention des modèles de simulation.

2.2.4 Vers une ingénierie de la simulation basé sur les modèles

La conception d'une simulation est une activité basée sur les modèles, qui suit un processus itératif. A partir de l'objectif d'utilisation de la simulation, un *modèle conceptuel* est conçu, indépendamment des choix d'implémentation. Puis à partir de ce modèle, le modèle de simulation (code exécutable) est développé. Une fois que le modèle de simulation est obtenu, une phase d'expérimentations permet de s'assurer que le système de simulation répond au besoin.

Aujourd'hui, les outils actuels de simulation permettent de s'affranchir de l'étape de programmation (du modèle de simulation) via l'utilisation du langage graphique de plus haut niveau qui permet de modéliser le modèle de simulation. On distingue ainsi deux grandes phases dans la conception d'une simulation : l'une de modélisation (obtention du modèle de simulation) et l'autre de simulation (où l'on exécute le modèle de simulation).

La phase de modélisation permettant d'obtenir du code peut être vue comme une activité basée sur les modèles : il s'agit de modéliser le comportement du système d'intérêt (via une représentation graphique), indépendamment de comment il sera exécuté, puis à partir de cette représentation, d'obtenir le modèle exécutable (le code). Dans une perspective « d'Ingénierie de la Simulation Dirigée par les Modèles », des similitudes entre IDM et le développement d'une simulation peuvent ainsi être mises en évidence. En effet, issus des phases d'analyse, de conception et d'implémentation, trois types de modèles sont considérés dans l'IDM, qui peuvent aussi s'appliquer à la simulation [Wagner et al., 2016] :

- les *modèles du domaine* (aussi appelés modèles conceptuels), qui sont indépendants de

la solution, décrivent le domaine du problème ou le système d'intérêt.
- les *modèles de conception* (de la simulation), qui sont indépendants de la plateforme, proposent une solution générique.
- les *modèles d'implémentation* (de la simulation), qui sont spécifiques à une plateforme, sont dérivés du modèle de conception, et peuvent directement être mappés à du code.

De récents travaux ont ainsi porté sur l'utilisation d'approches issues de l'IDM pour la simulation, un aperçu de ces travaux est notamment proposé par [Cetinkaya and Verbraeck, 2011].

La simulation étant une activité basée sur les modèles, l'utilisation des techniques de l'IDM, en particulier au travers des transformations de modèles pourrait s'avérer intéressante afin de faciliter l'obtention du modèle de simulation (en tant qu'exécutable), et de faciliter son déploiement dans un contexte industriel pour des non experts.

2.3 Bilan sur l'apport de la simulation

La simulation informatique est un moyen pour conduire des expérimentations, en modélisant la dynamique d'un système. Elle peut ainsi permettre de tester un système de contrôle-commande sans pour autant disposer d'une maquette physique du système à commander. De façon générale, on distingue trois approches de test : Model-in-the loop (qui permet de tester un modèle de la commande), Software-in-the-loop (qui permet de tester la partie logicielle de système de commande) et Hardware-in-the-loop (qui permet de tester le système de commande réel). Ces deux dernières approches sont notamment utilisées dans le cadre du *virtual commissioning* (phase de tests virtuels effectués avant la phase de tests réels). La simulation semble être un moyen de permettre la mise en place de tests de système de contrôle-commande avant la construction du système physique, notamment via le *virtual commissioning*, ou même pour aller plus loin, en parallèle des activités de conception du système de contrôle-commande.

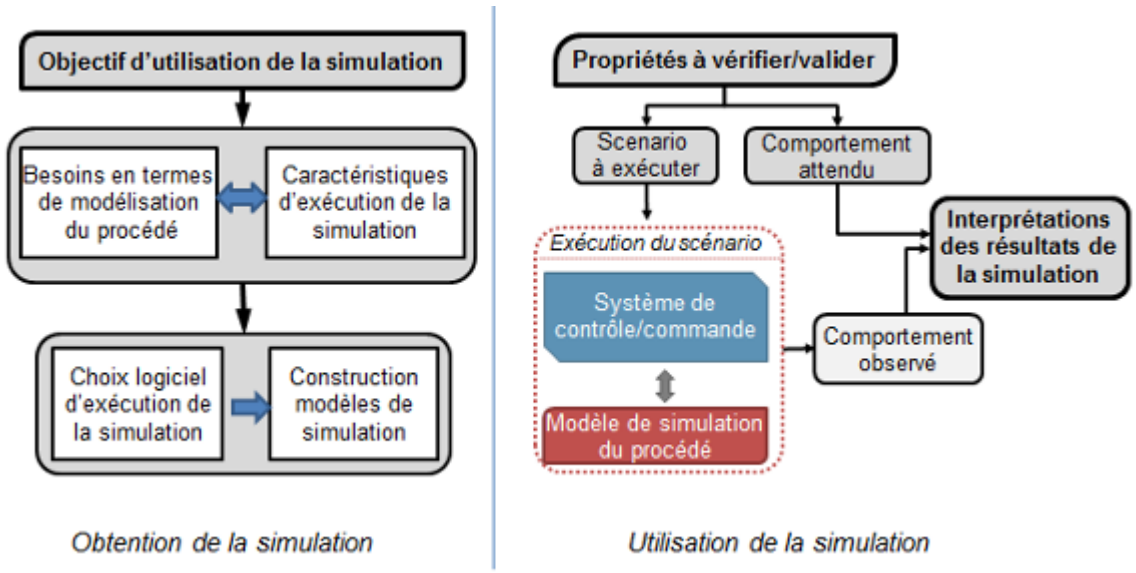


FIGURE 2.7 – Obtention et utilisation de la simulation

Cependant, l'obtention de la simulation requiert la mise en oeuvre d'un projet de simulation (tel un projet de conception). Cela demande des connaissances approfondies en simulation, et du procédé à modéliser. De plus, l'obtention de modèles de simulation reposant sur un problème de modélisation, il faut être en mesure de bien définir l'objectif d'utilisation (Figure 2.7).

Lors de la modélisation, il faut ainsi prendre en compte le niveau de détail requis pour les besoins d'utilisation et prendre en compte les caractéristiques d'exécution nécessaires (le choix d'un langage de simulation, et le choix d'une plateforme d'exécution par exemple). D'autre part, il faut aussi prendre en compte les phénomènes et informations que l'on doit pouvoir observer (afin de pouvoir interpréter les résultats de simulation), ce qui dépend des vérifications que l'on veut effectuer.

L'utilisation de la simulation pour conduire des tests répond donc à un besoin, qu'il faut être en mesure d'exprimer pour pouvoir modéliser le procédé. L'obtention manuelle étant coûteuse en temps, un intérêt est porté sur son obtention automatisée, d'autant plus qu'il s'agit d'une activité basée sur les modèles.

3

Synthèse : Problématique et Orientations

Sommaire

3.1 Objectifs des travaux

Afin d'améliorer les méthodes et outils de conception de systèmes de conduite de procédés, la mise en place de vérifications et validations tout au long de la conception est essentielle. D'une part pour corriger les erreurs de conception le plus tôt, à moindre coût (en terme d'effort, de temps et d'argent). Et d'autre part, afin de pouvoir intégrer l'utilisateur final dans la conception (cadre de conception participative). Pour ce faire, il faut disposer d'un prototype du système.

Dans un contexte d'ingénierie simultanée avec des cycles courts et itératifs, et de démarche de génération automatisée de programmes de commande et d'interfaces de supervision, la partie logicielle du système de conduite peut être disponible très tôt (même si incomplète). L'introduction de tests peut donc se faire relativement tôt et de façon progressive, néanmoins encore faut-il disposer d'un prototype virtuel du procédé.

De fait, **l'objectif principal de nos travaux est de permettre l'intégration de tests dès le début de la conception, en facilitant l'obtention du prototype virtuel d'un procédé**. Nous nous proposons de l'appliquer aux procédés de type gestion de fluide.

Toutefois, avant de confronter l'utilisateur final à la solution retenue pour le système de conduite, il convient d'avoir vérifié la solution par rapport aux spécifications (aspects *aide à la décision et à la surveillance*, *fonctions de haut niveau*, *adaptation du système face aux aléas*).

L'objectif secondaire de nos travaux concerne ainsi la mise en place de vérifications au plus tôt dans un flot de génération automatisé de programmes de commande et d'interfaces de supervision. Les travaux se déroulant au travers d'une thèse CIFRE, nous nous proposons de l'appliquer dans le cadre de la démarche Anaxagore.

3.2 Problématique et verrous identifiés

Comme évoqué précédemment, nous cherchons à mettre en place une démarche de vérification d'un système de conduite de procédé de gestion de fluide, via l'utilisation d'un prototype virtuel du procédé.

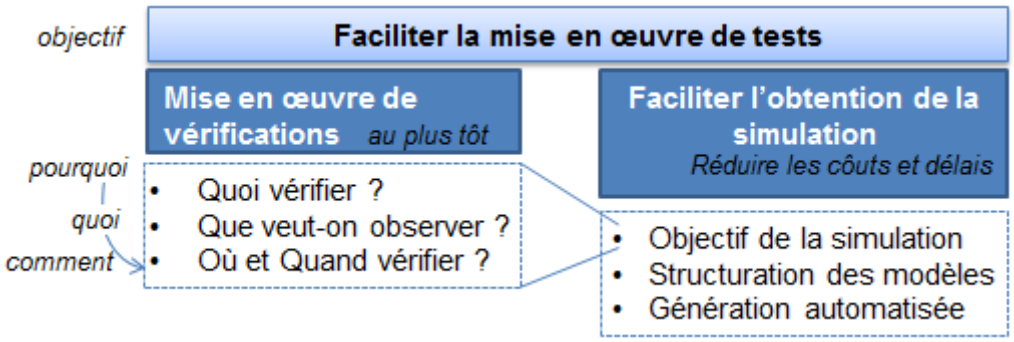


FIGURE 3.1 – Synoptique des travaux

Avant de s'intéresser à faciliter l'obtention de la simulation (Figure 3.1), il faut définir l'*objectif d'utilisation de la simulation.*

- Il faut donc être en mesure d'exprimer ce que l'on veut vérifier, et quelles observations doit-on faire sur le procédé pour pouvoir statuer sur la/les propriété(s).
- Il s'agit également de définir la place des vérifications dans le flot de conception au plus tôt, ce qui permettra de choisir l'approche de test adéquate (MiL ou SiL).

Il est à noter que ce dernier point n'est pas un verrou en soi, il dépend de ce que l'on veut vérifier et de la disponibilité des informations nécessaires pour effectuer les vérifications.

Une fois, l'objectif d'utilisation de la simulation défini, nous pouvons nous intéresser au problème de modélisation du procédé et de son obtention.

- Il s'agit de définir les principes de modélisation, outre le niveau de détail approprié, cela consiste à définir une stratégie d'organisation des modèles de simulation.
- Puis à partir de la structuration des modèles, nous nous intéressons à l'obtention des modèles de simulation, au travers d'un flot de génération automatisée.

Par rapport à notre problématique, nous avons identifiés trois verrous (Figure 3.2), nous les présentons plus en détail dans la suite de la section.

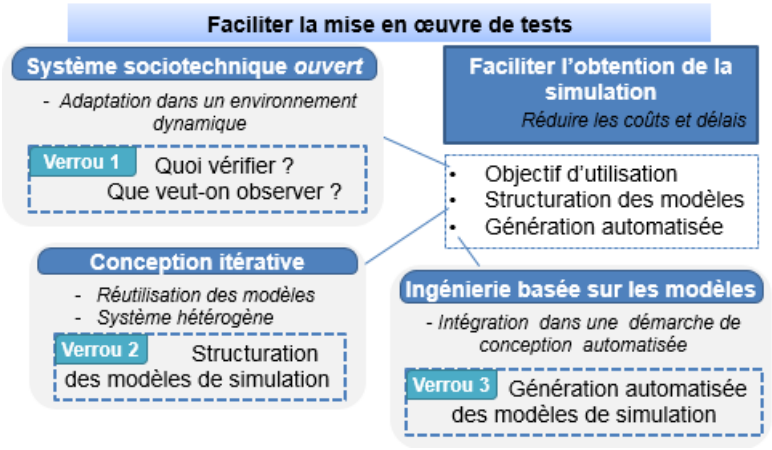


FIGURE 3.2 – Verrous identifiés

3.2.1 Verrou 1 : Formalisation des propriétés

Que veut-on vérifier ? Que doit-on observer ? La réponse à ces questions se pose bien sûr dans le cadre de l'intégration de techniques de vérification au sein de la démarche Anaxagore, mais pas uniquement. De manière plus générale ces questions se posent dès lors que l'on veut effectuer des tests (Figure 3.3). La réponse à ces questions permet de définir le contexte intentionnel, le *pourquoi* de la mise en oeuvre de tests et permet d'exprimer l'objectif de la simulation.

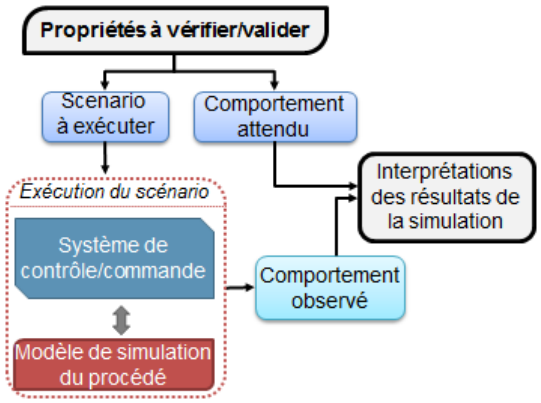


FIGURE 3.3 – Principe d'utilisation de la simulation

Outre l'intérêt de connaître les besoins d'utilisation pour déterminer les caractéristiques à prendre en compte dans la modélisation du comportement du procédé, au travers de ces questions on s'intéresse à l'activité de la personne qui va réaliser les tests, la simulation étant un instrument dans son activité. Répondre à ces questions revient à formaliser les propriétés à vérifier dans le contexte.

En effet, le système considéré étant un système sociotechnique ouvert, il évolue dans un contexte dynamique auquel il s'adapte. Le système disposant de différents moyens qui peuvent être mis en oeuvre pour atteindre ses objectifs, pour garantir la sûreté de fonctionnement il faut

donc être en mesure de pouvoir exprimer le comportement attendu par rapport à la situation. Cependant, comme le système dispose de plusieurs possibilités pour rendre un service, il faut pouvoir identifier quels moyens sont mis en oeuvre et à quel moment.

Ainsi, pour un système sociotechnique ouvert, afin d'étudier son comportement dans un environnement dynamique il est nécessaire de pouvoir exprimer les propriétés dans le contexte. La question est donc : *Comment formaliser les propriétés dans le contexte ?.*

3.2.2 Verrou 2 : Structuration des modèles de simulation

Ce deuxième verrou concerne la capacité de générer des modèles de simulation selon les différents besoins de modélisation requis. En effet, on vise une démarche de génération de modèles de simulation qui puisse être réutilisée tout au long de la conception, malgré des besoins en termes de modélisation potentiellement très variés. Ainsi, de nombreux modèles de simulation doivent potentiellement être générés, avec potentiellement des plateformes d'exécutions (ou logiciel de simulation) différents.

De plus, du fait de la nature du procédé (système de gestion de fluide), il ne s'agit pas de modéliser uniquement la partie opérative du procédé, le comportement du fluide doit aussi être représenté. Ce procédé est un système hétérogène, avec une partie opérative dont l'évolution du comportement est plutôt à événements discrets, et l'évolution du comportement du fluide qui est continue. Il s'agit donc de proposer une stratégie de modélisation qui permette de modéliser le système hétérogène, tout en permettant de s'adapter à l'évolution des besoins de représentation.

En d'autres termes, on cherche à proposer une structuration des modèles de simulation, qui soit adaptée à une activité de modélisation du système hétérogène et qui puisse être réutilisée tout au long de la conception en vue de réaliser des tests. La question est donc : *Comment structurer les modèles de simulation pour permettre la réutilisation ?*

3.2.3 Verrou 3 : Génération automatisée des modèles de simulation

Le dernier verrou identifié concerne la démarche de génération automatisée des modèles de simulation. On cherche ici à automatiser l'obtention des modèles de simulation, dans le cadre d'une ingénierie simultanée et dirigée par les modèles. Comme évoqué au travers du verrou 2, on vise une obtention de la simulation en parallèle des activités de conception, tout au long de cette dernière.

Outre les problèmes de modélisation hétérogène et de réutilisation, il s'agit ici d'intégrer une démarche de génération automatique de la simulation au sein d'un flot de conception basé sur l'IDM. La question est donc : *Comment intégrer la génération automatisée des modèles de simulation dans un flot de conception automatisé de contrôle-commande ?*

3.3 Schéma directeur des travaux

Le plan de lecture des travaux s'articule en deux parties : une partie propositions théoriques (Figure 3.4) et une partie applications (Figure 3.5).

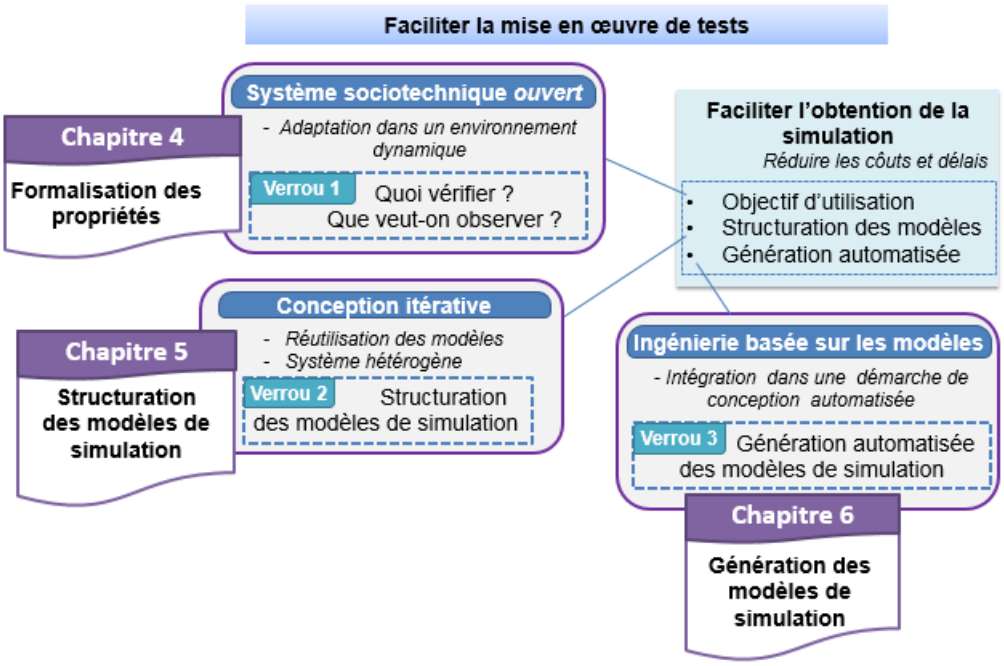


FIGURE 3.4 – Plan de lecture de la partie Propositions Théoriques

La première partie de propositions théoriques se compose de trois chapitres. Bien qu'il existe des liens entre les verrous identifiés, chacun d'entre eux traite des problèmes spécifiques. Pour plus de lisibilité et d'intelligibilité, chaque verrou est traité dans un chapitre dédié, contenant l'état de l'art associé ainsi que nos contributions théoriques.

Puis, nous présentons l'application de nos propositions, au travers d'une preuve de concept concernant l'obtention automatisée des modèles de simulation, et au travers d'une preuve d'usage concernant la mise en oeuvre de la démarche de vérification. La preuve de concept et la preuve d'usage font chacune l'objet d'un chapitre.

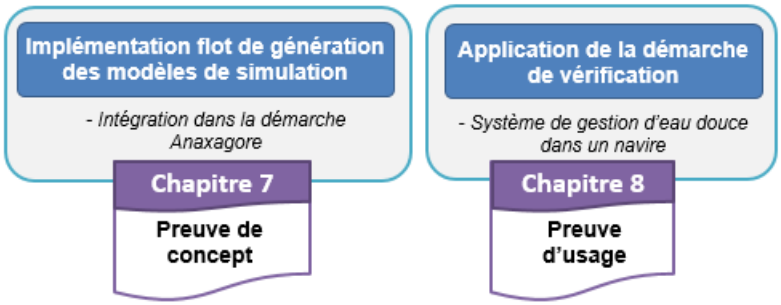


FIGURE 3.5 – Plan de lecture de la partie Applications

Partie B -

Propositions théoriques

Cette deuxième partie du manuscrit vise à lever les verrous précédemment identifiés, afin de pouvoir faciliter la mise en oeuvre de tests des systèmes de conduite de procédés tout au long de la conception. La levée de ces verrous se fera via trois chapitres, chacun d'entre eux adressant un des verrous. Chaque chapitre contient un état de l'art associé au verrou traité, permettant ainsi de construire nos propositions.

Le Chapitre 4 a pour objectif de faciliter l'interprétation des résultats issus de la simulation. Il s'agit ici de s'attaquer au verrou concernant la formalisation des propriétés, et notamment celles portant sur l'adaptation du système dans un environnement dynamique. Pour cela, un formalisme est proposé, adapté au cadre des systèmes sociotechniques, et permettant de modéliser les propriétés dans le contexte.

Les deux chapitres suivant visent quant à eux à faciliter l'obtention de la simulation, pour pouvoir réaliser des tests de façon continue en parallèle de la conception.

Le Chapitre 5 a pour objectif de proposer une structuration des modèles de simulation pour permettre leur réutilisation, malgré des besoins de modélisation différents selon le niveau où l'on se situe dans le processus de conception. Il s'agit, dans le cadre de la modélisation d'un système hétérogène, de déterminer *comment structurer les modèles de simulation pour permettre la réutilisation.*

Ceci permettra au travers du Chapitre 6, de s'intéresser à l'obtention automatisée des modèles de simulation. L'objectif de ce chapitre est ainsi de proposer un flot de génération automatisé qui puisse être intégré dans un flot de conception automatisé de contrôle-commande.

4

Formalisation des propriétés à vérifier par simulation

Sommaire

4.1 Introduction

Améliorer la conception de systèmes sociotechniques (ouverts) consiste, entre autres, à limiter la détection tardive d'erreurs, réduisant ainsi le temps et les coûts de conception. Garantir la qualité de systèmes de conduite de procédés nécessite ainsi la mise en oeuvre de tests dès que possible. Un intérêt est porté sur la mise en oeuvre de tests au fur et à mesure que la conception avance, en utilisant des techniques de simulation pour obtenir un prototype virtuel du procédé.

Dès les premières phases de conception, lorsque le système physique n'est pas figé, la simulation peut être utilisée pour effectuer des vérifications fonctionnelles/comportementales. On peut, par exemple, chercher à vérifier, comme dans le cas de modèles de commande [Chapurlat, 2007] :

- que quelque chose de mauvais ne doit pas se produire (propriété de sûreté) ;
- que quelque chose de bon doit finalement arriver (propriété de vivacité) ;
- qu'une situation ou un état donné seront atteints tôt ou tard (propriété d'atteignabilité) ;
- que quelque chose doit toujours arriver, ou que son contraire ne doit jamais arriver (propriété de sécurité).

Toutefois, l'obtention des modèles de simulation du procédé nécessite, outre la modélisation des entrées/sorties avec la partie commande, de représenter le comportement du procédé (au moins en termes de logique). Le niveau de détail de la modélisation dépend des tests que l'on veut effectuer, cela demande donc de définir au préalable ce que l'on veut vérifier.

Le système de conduite est un système de contrôle-commande. Il vise à restreindre l'ensemble des évolutions possibles du procédé (système physique) via la partie opérative, afin que le procédé puisse, malgré les évolutions de l'environnement, continuer à accomplir ses objectifs. Le comportement du système physique est ainsi contraint à la fois par les lois de la physique et par le système de contrôle-commande (caractère intentionnel). Vérifier le bon fonctionnement du système de conduite revient de façon générale à se demander si ce dernier est capable d'emmener/garder le système physique dans un état qui fournit les services requis par rapport à la situation (qui est dynamique). En effet, le caractère adaptable du système est une caractéristique importante, comme évoqué dans le premier chapitre. S'agissant d'un système sociotechnique, garantir les aspects de sûreté de fonctionnement et de résilience est ainsi essentiel.

Trois catégories de vérifications[1], liées à la nature sociotechnique du système ont été identifiées : celles concernant *l'aide à la décision et à la surveillance*, celles concernant *les fonctions de haut niveau* qui doivent être mises en oeuvre par la partie technique du système de conduite, et enfin celles concernant *l'adaptation du système face aux aléas*. Au travers de ces catégories, les vérifications peuvent être appliquées à différents niveaux de décomposition du système de contrôle-commande (granularité) :

- au niveau *composant* (c'est-à-dire un élément du système)
- au niveau d'une *fonction* (c'est à dire un groupe d'éléments qui peut être utilisé pour atteindre un objectif du système)
- au niveau *système* (le système dans son ensemble).

Ainsi, selon le niveau de formalisation des propriétés à vérifier, le niveau d'abstraction dans la modélisation du procédé peut être très différent, notamment en termes d'observables (les informations devant apparaître dans les résultats de simulation).

Outre l'objectif d'utilisation de la simulation, l'expression des propriétés à vérifier est également déterminante dans la mise en oeuvre des tests (cf Figure 4.1). C'est à partir de celles-ci que sont définis les scénarios à exécuter ainsi que les comportements attendus associés. La formalisation des propriétés, au travers de la spécification du comportement attendu, aura ainsi un impact significatif sur la phase d'analyse des résultats de la simulation. Selon comment on modélise le comportement attendu, la comparaison avec le comportement obtenu lors de la simulation peut s'avérer fastidieuse, et potentiellement source d'erreurs. En effet, interpréter les résultats d'une simulation n'est pas toujours évident. Il existe un écart (potentiellement important) entre les données "brutes" que l'on obtient suite à l'exécution de la simulation, et une représentation intelligible des données d'intérêt. En effet, les commandes envoyées au procédé sont de bas niveau, or si l'on cherche à vérifier des propriétés à l'échelle du système ou de fonctions de haut niveau, il peut y avoir un fossé entre les résultats de la simulation (de

1. Voir au Chapitre I section 3.2.

bas niveau), et le comportement attendu tel qu'il a été défini (de plus haut niveau), menant potentiellement à des erreurs d'interprétation.

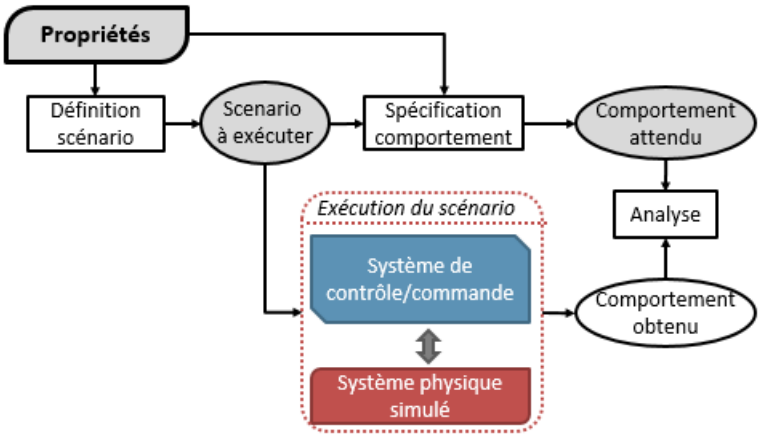


FIGURE 4.1 – Principe d'une vérification par simulation

Par exemple, si l'on cherche à vérifier la délivrance d'un service malgré l'occurrence d'aléas, la continuité de service peut être spécifiée par des observables à différents niveau de granularité. Si l'on prend le cas d'un transfert entre deux soutes, les observables peuvent être que le niveau de la soute de départ baisse tandis que celui de la soute d'arrivée augmente. Toutefois, selon l'architecture du système, ces observables peuvent également correspondre à un approvisionnement de la soute d'arrivée et à une distribution à partir de la soute de départ vers un autre sous-système. En effet, les systèmes de gestion de fluide peuvent délivrer plusieurs services simultanément. Ainsi, les observables doivent être définis en tenant compte du contexte, pour faciliter l'analyse de la simulation. De plus, le système disposant de plusieurs moyens pour atteindre les objectifs et évoluant dans un contexte dynamique, vérifier le caractère reconfigurable du procédé est important. Il s'agit en particulier de regarder comment le système fait face aux aléas pour assurer la disponibilité du service. La phase d'analyse des résultats de la simulation sera ainsi plus ou moins importante, selon le nombre de reconfigurations possibles (ou admissibles) du système d'une part, et selon la formalisation des propriétés et du comportement attendu d'autre part.

Il apparaît ainsi un écart entre la formalisation des propriétés et les résultats issus de la simulation. De ce fait, un résultat de simulation pourra satisfaire ou non une propriété pour une situation donnée. Il est donc important de contextualiser l'expression des propriétés, et ce à différent niveaux de granularité pour s'adapter aux différentes vérifications envisagées. Ceci correspond au premier verrou identifié dans le chapitre 3.

Suite à l'état de l'art réalisé dans la section 4.2, nous proposons dans la section 4.3 un formalisme de modélisation des propriétés.

4.2 Etat de l'art

Plusieurs travaux se sont intéressés à la spécification des propriétés, que ce soit dans les phases préliminaires de conception, dans le cadre de l'ingénierie des exigences [Dobre et al., 2010],

ou par exemple pour la mise en oeuvre de vérifications par méthodes formelles (cf la Systématic Mapping réalisée dans les travaux de [Mesli Kesraoui, 2017]). Toutefois, ils ne sont pas adaptés pour notre besoin, l'écart entre ces modélisations et le traitement des résultats de simulation étant trop important. Nous avons besoin d'un formalisme pour la modélisation des propriétés adapté au cas de systèmes sociotechniques ouverts, afin de faciliter l'interprétation des résultats.

Pour tester le caractère reconfigurable d'un système, il est nécessaire de modéliser la structure, le comportement et/ou les fonctions par rapport au contexte. Nous ne présenterons ici que les concepts utiles à la construction de notre proposition. En premier lieu, nous nous intéressons aux aspects du système à modéliser dans les phases préliminaires de la conception. Nous explorerons ensuite les domaines de l'ingénierie spécifiques aux caractéristiques sociotechniques et reconfigurables.

4.2.1 Les aspects du système à modéliser en conception préliminaire

En conception préliminaire, les exigences peuvent être capturées à travers différents points de vue : *fonctionnel*, *comportemental*, *structurel* ou bien en tant que *contrainte*. Gero a proposé un modèle de la conception, le *Function-Behavior-Structure* (FBS) [Gero, 1990], dans lequel des connexions sont faites entre les différents concepts centraux que sont les *fonctions*, le *comportement* et la *structure* du système considéré par les concepteurs. Les **fonctions** décrivent ce pour quoi il est fait, les **comportements** décrivent ce qu'il fait, et la **structure** décrit ce qu'il est (les relations entre les composants). Ce modèle a ensuite été étendu pour représenter la conception dans un monde dynamique [Gero and Kannengiesser, 2004].

Dans le cadre de l'ingénierie dirigée par les modèles, [Christophe et al., 2010] a adapté le modèle FBS, pour inclure la modélisation des exigences à partir desquelles les fonctions sont dérivées, en proposant le modèle *Requirement-Function-Behavior-Structure* (RFBS). De plus, une distinction est faite entre une **fonction de service** (ce que le système fournit à son environnement, en utilisant une modélisation de type boîte noire) et une **fonction technique** (qui exprime comment le système réalise le service, approche de type boîte blanche).

En Ingénierie Système, l'INCOSE promeut l'utilisation de langage de modélisation tel que SysML [OMG SysML, 2015], qui permet la capture des exigences, des fonctions, de la structure et du comportement à travers différents modèles. Cependant, pour pouvoir dériver les scénarios de test et le comportement attendu à partir de l'expression des propriétés, les informations doivent être représentées dans le contexte. Ceci est d'autant plus vrai, puisque l'on s'intéresse au caractère adaptable du système. Il nous semble ainsi intéressant de pouvoir représenter l'ensemble des informations nécessaires dans un même modèle.

4.2.2 Les contributions issues de l'ingénierie des systèmes sociotechniques

Les systèmes sociotechniques étant caractérisés par une interaction complexe entre les humains, les machines, et le contexte (les aspects environnementaux du système de travail), les méthodes de conception doivent prendre en compte ces facteurs [Baxter and Sommerville, 2011].

Selon [Naikar et al., 2005], l'approche *Cognitive Work Analysis* (CWA) reconnaît deux caractéristiques essentielles de tels systèmes, à savoir qu'ils sont dynamiques et qu'ils ont un haut niveau d'automatisation. Le CWA [Rasmussen et al., 1994, Vicente, 1999] est notamment utilisé pour l'analyse, la conception et l'évaluation de systèmes sociotechniques complexes. Cette approche se focalise sur *comment le "travail"* ***peut*** *être effectué*. Pour cela, il faut notamment étudier les différentes contraintes qui vont restreindre le comportement du "travailleur", selon plusieurs dimensions (phases) d'analyse.

La première phase de cette approche est l'analyse du domaine de travail, *Work Domain Analysis* (WDA), dans laquelle l'espace d'abstraction et décomposition (*abstraction-decomposition space*, en anglais) est utilisé pour modéliser le contexte de travail de l'opérateur [Naikar et al., 2005]. Le but de cette analyse est de représenter les contraintes liées à l'environnement de travail. Cet environnement peut se décomposer en un contexte physique et un contexte intentionnel (permettant de définir un espace problème/solution). Ainsi, les contraintes liées au contexte physique concernent les éléments disponibles dans le système de travail, leur capacités fonctionnelles ainsi que leur limitations. Les contraintes intentionnelles concernent les objectifs que le système doit remplir, les critères à satisfaire ainsi que les fonctions à effectuer pour atteindre les objectifs. Le contexte intentionnel explique pourquoi le système existe, quelle est sa mission, alors que le contexte physique explique comment le système peut remplir sa mission.

TABLE 4.1 – Les cinq niveaux de la hierarchie d'abstraction [Naikar et al., 2005]

Hiérarchie d'Abstraction	*Description*
Objectifs Fonctionnels	Objectifs du système de travail et contraintes externes sur son fonctionnement
Valeurs et Mesures Prioritaires	Critères utilisés par le système de travail pour mesurer son progrès par rapport aux objectifs fonctionnels
Fonctions reliées aux Objectifs	Fonctions générales du système de travail qui sont nécessaires pour accomplir les objectifs fonctionnels
Processus reliés aux Fonctions	Capacités et limitations fonctionnelles des objets physiques dans le système de travail qui permettent la réalisation des fonctions reliées aux processus
Objets Physiques	Objets physiques dans le système de travail qui fournissent les processus reliés aux objets

L'analyse du domaine de travail se fait suivant un axe d'abstraction, constitué de cinq niveaux (cf Table 4.1). Les trois premiers niveaux d'abstraction (les plus abstraits) modélisent les propriétés intentionnelles de l'espace de problème. Cela définit le *pourquoi* du système. Les deux niveaux les plus bas modélisent, quant à eux, les propriétés physiques de l'espace de problème. Cela définit les ressources disponibles et leurs fonctionnalités pour obtenir le comportement du système, c'est à dire le *comment*.

L'espace de décomposition permet de représenter l'espace de problème à différents niveaux de détails (le système entier, les sous-systèmes, et les composants). L'utilisation de l'espace d'abstraction et de décomposition semble correspondre à notre besoin de représentation. Cela

permet de capturer au travers d'un même modèle, les objectifs du système, les moyens permettant de les réaliser et les critères d'évaluation à différents niveau de décomposition. Toutefois, la formalisation des informations à l'intérieur du modèle est laissée à charge de l'analyste.

4.2.3 Les contributions issues des systèmes manufacturiers flexibles

Dans le cadre des systèmes manufacturiers flexibles (FMS), les notions de fonction et d'opérations ont été utilisées pour capturer les caractéristiques de ces systèmes [Berruet et al., 1999].

Définition 4.1 (fonction et opération)
Une *fonction* est un service délivré par un système.
Une *opération* est une fonction mise en oeuvre par une ressource lorsque la ressource est considérée comme un composant du système global.

Ainsi, une fonction (fonction de service) est mise en oeuvre par un ensemble d'opérations (fonctions techniques). Cependant, selon [Berruet et al., 1999] une fonction est mise en oeuvre par une seule opération à un instant donné. Dans les travaux de [Toguyéni et al., 2003], la notion de fonction générique et de fonction contextuelle ont été introduites. Les **fonctions génériques** d'un élément de l'usine sont définies sans faire référence au comportement de l'usine là où cet élément est utilisé. Au contraire, les **fonctions contextuelles** sont introduites par le concepteur d'un système pour adapter ou pour composer des fonctions génériques données par les constituants, en réponse aux exigences du système modélisé. Cette notion peut aussi être étendue aux opérations [Berruet et al., 2005].

Un système reconfigurable est capable d'évoluer suivant les exigences de qualité de services dans un contexte variable, en changeant sa configuration [De Lamotte et al., 2006]. La configuration décrit comment les ressources (les entités du système) sont utilisées pour atteindre un objectif. [Kanso et al., 2009] a par ailleurs introduit la notion de configuration minimale et maximale.

Définition 4.2 (configuration minimale et maximale)
La configuration *maximale* est l'ensemble des opérations potentielles qui permettent de répondre à la demande. La configuration *minimale* est l'ensemble des opérations juste nécessaires pour répondre à la demande.

De plus, la reconfiguration est définie comme "une phase de transition entre deux configurations" (entre la requête accomplie et la suivante, ou lorsque la configuration courante ne permet plus de satisfaire les exigences suite à l'occurrence d'aléas). Une reconfiguration est ainsi le passage d'une configuration à une autre.

Cette notion de *configuration* est d'autant plus intéressante qu'elle permet de définir l'état du système comme l'ensemble des opérations courantes mises en oeuvre. A travers cette notion de configuration (comme l'ensemble d'opérations mises en oeuvre), la reconfiguration peut être définie comme le changement d'un ensemble d'opérations courantes mises en oeuvre vers un autre ensemble d'opérations mises en oeuvre.

4.2.4 Vers une modélisation des propriétés

Pour mener à bien des vérifications fonctionnelles/comportementales d'un système de contrôle/commande dans un système sociotechnique reconfigurable, les exigences doivent être modélisées en contexte. Un tour d'horizon des représentations de la connaissance a été réalisé dans ce sens. La notion de *configuration* semble être adéquate pour décrire l'état du système, et qui plus est, pour indiquer parmi l'ensemble des possibilités laquelle est mise en oeuvre. Ainsi, un intérêt dans la réalisation de vérifications peut être de répondre à la question « La configuration est-elle correcte, par rapport aux objectifs et à la situation ? ».

En outre, si nous considérons différentes limites du système (c'est à dire différents niveaux de vérification), nous pouvons distinguer la *configuration d'une fonction* (reliée à une demande de haut niveau) et la *configuration du système*.

Définition 4.3 (configuration d'une fonction et configuration du système)
Une configuration *d'une fonction* correspond à une possibilité de mise en oeuvre d'une requête de haut niveau par un ensemble d'opérations.
La configuration *du système*, à un instant donné, est l'ensemble des opérations qui sont implémentées.

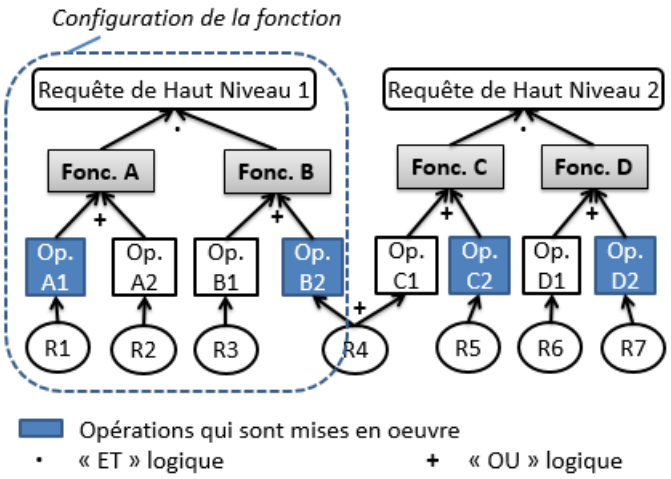


FIGURE 4.2 – Notion de configuration d'une fonction

Par exemple, sur la Figure 4.2, une configuration de la fonction liée à la *Requête de Haut Niveau 1* correspond aux *Opérations A1* et *B2* en utilisant les ressources R1 et R4. Une autre configuration correspond aux *Opérations A2* et *B1* en utilisant les ressources R2 et R3. La configuration du système sur la Figure 4.2 correspond aux opérations qui sont représentées en bleu, à savoir les *Opérations A1, B2, C2* et *D2* en utilisant les ressources R1, R4, R5 et R7. Ceci correspond à l'ensemble des configurations courantes des fonctions (liées aux reqûetes 1 et 2). Ainsi, la configuration du système peut être vue comme l'ensemble des configurations de fonctions qui sont implémentées (à l'instant considéré).

Les vérifications réalisées au niveau d'une fonction visent ainsi à répondre à la question « La configuration de la fonction est-elle correcte par rapport à l'objectif et à la situation ? ». Si l'on s'intéresse plus particulièrement au caractère reconfigurable du système, la question

devient « La *nouvelle* configuration de la fonction est-elle correcte par rapport à l'objectif et à la situation ». Au niveau système, les vérifications visent à répondre à la question « La nouvelle configuration du système est-elle correcte par rapport à l'objectif et à la situation ». D'autre part, selon l'objectif et la situation, une fonction de service de haut niveau du procédé peut être implémentée, soit par un ensemble d'opérations, soit par une seule opération. De ce fait, représenter les configurations minimales et maximales serait intéressant.

A partir des différents points de vue, une synthèse de la notion de *fonction* est proposée Table 4.2. Il est à noter que le niveau d'abstraction *fonctions reliées aux objectifs* peut faire référence soit à des fonctions *génériques*, soit à des fonctions *contextuelles*, selon le niveau de décomposition où l'on se place (système, sous-système, composant). Typiquement, au niveau de décomposition Composant, les fonctions sont génériques car elles ne font pas référence au comportement du système ou sous-système.

TABLE 4.2 – Synthèse sur la notion de *fonction*

Aspects du système à modéliser	*Contribution des systèmes sociotechniques*	*Contribution des systèmes manufacturiers flexibles*
Fonction	Objectifs fonctionnels	Requêtes/Mission
	Valeurs et mesures prioritaires	Qualité de service
	Fonctions reliées aux objectifs	Fonctions
Comportement	Processus reliés aux objets	Opérations
Structure	Objets physiques	Ressources

Pour mettre oeuvre des vérifications par simulation, nous proposons donc de nous appuyer sur la *Hiérarchie d'Abstraction* pour représenter les propriétés, et d'utiliser les concepts de *fonction* et *opération* pour formaliser les informations, et ce à différents niveaux de décomposition (composant, fonction, système).

4.3 Proposition d'un formalisme de modélisation

Le but de notre proposition de modélisation est de pouvoir représenter les propriétés comme une ou plusieurs configurations à travers une analyse fonctionnelle dans laquelle les fonctions sont contextualisées et reliées aux ressources. Dans un premier temps nous présenterons la proposition de modélisation contextualisée, puis dans un second temps nous présenterons la méthodologie associée permettant de faciliter l'analyse des résultats de la simulation.

4.3.1 Une modélisation multi-niveau contextualisée

Nous appliquerons notre proposition de modélisation à des systèmes de conduite de procédés de type gestion de fluide. Dans un système de gestion de fluide, le procédé est constitué d'une partie opérative (vannes, pompes...) qui peuvent agir sur le fluide, et des éléments de tuyauteries (soutes ...). L'utilisateur final (opérateur) peut interagir avec le procédé à travers l'IHM de supervision en envoyant des requêtes de bas niveau (par exemple : ouverture de

telle vanne, démarrage de telle pompe...) ou bien de haut niveau (par exemple : un transfert entre deux soutes). En conséquence, le système de contrôle/commande restreint l'ensemble des évolutions possibles du procédé suivant les requêtes.

Nous considérerons en guise d'exemple illustratif le cas d'étude représenté sur la Figure 4.3. Il s'agit d'un système composé d'éléments qui permettent de produire de l'eau douce à partir d'eau de mer (Osmoseurs), de soutes pour stocker l'eau produite, de vannes motorisées pour guider le fluide dans le réseau de tuyauterie, et de pompes pour faire circuler le fluide. Afin de garantir la potabilité de l'eau, un module de chloration est utilisé pour traiter l'eau.

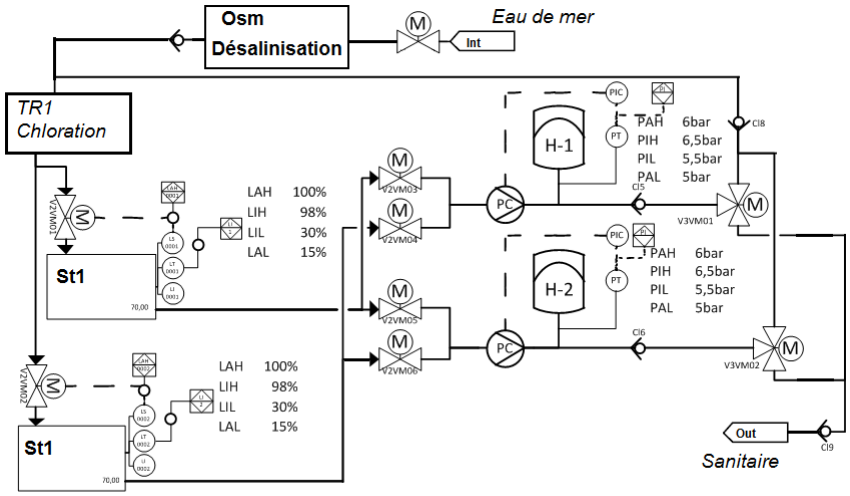


FIGURE 4.3 – Exemple de système d'eau douce sanitaire

Avant d'appliquer le cadre de modélisation aux différents niveaux de vérification (décomposition), il nous faut définir les fonctions et opérations pour les procédés de type gestion de fluide. Les services de base qui doivent être accomplis par un tel système sont décrits ci-dessous, sous forme de fonctions génériques. Ces fonctions génériques peuvent s'appuyer sur une taxonomie de l'ingénierie système (par exemple, [Hirtz et al., 2002]).

Définition 4.4 (fonction de Générateur)
Une fonction de **Générateur** génère un mouvement du fluide (en imposant une pression ou un débit). Une telle fonction est implémentée par une opération de *Générateur par une ressource* (par exemple une pompe) qui génère une pression ou un débit.

Définition 4.5 (fonction de Transformation)
Une fonction de **Transformation** modifie les caractéristiques du fluide (par exemple : traitement de l'eau, désalinisation de l'eau de mer). Elle est mise en oeuvre par une opération de *Transformation réalisée par une ressource* qui modifie une ou plusieurs propriétés du fluide.

Définition 4.6 (fonction de Stockage)
Une fonction de **Stockage** garde le fluide dans une zone donnée. Différents types d'opérations peuvent implémenter une telle fonction. Une opération de *Stockage passif* est utilisée lorsque la ressource réalisant l'opération n'est pas commandée (cas d'un réservoir ou d'une soute). Au contraire, une opération de *Stockage Actif* nécessite de commander la ressource (fermeture d'une vanne par exemple).

Définition 4.7 (fonction de Libre Passage)
Une fonction de **Libre Passage** permet le passage du fluide en un lieu donné. Elle est implémentée par une opération de *Libre Passage qui est réalisée par une ressource* qui permet au fluide de passer (comme une vanne ouverte). Cette opération est similaire à une opération de *Stockage Actif* mais avec une durée de stockage nulle.

Définition 4.8 (fonction de Contrôle)
Une fonction de **Contrôle** mesure certains paramètres du fluide. Cette fonction est implémentée par une opération de *Contrôle réalisé par un capteur de mesure.*

Les fonctions génériques et les opérations les implémentant ayant été définies, nous pouvons appliquer le cadre de modélisation aux différents niveaux de décomposition du procédé.

4.3.1.1 Modélisation au niveau Composant

Dans ce niveau de modélisation, il s'agit de représenter, pour chaque type de composant du système, les fonctions de services (fonctionnalités) et les fonctions techniques qui peuvent être réalisées par ce composant. De plus, dans le niveau d'abstraction *Valeurs et mesures prioritaires*, le(s) critère(s) associé(s) à la délivrance du service est représenté. Par exemple, si l'objectif fonctionnel d'une vanne est d'empêcher le passage du fluide,alors celle-ci doit être en position fermée.

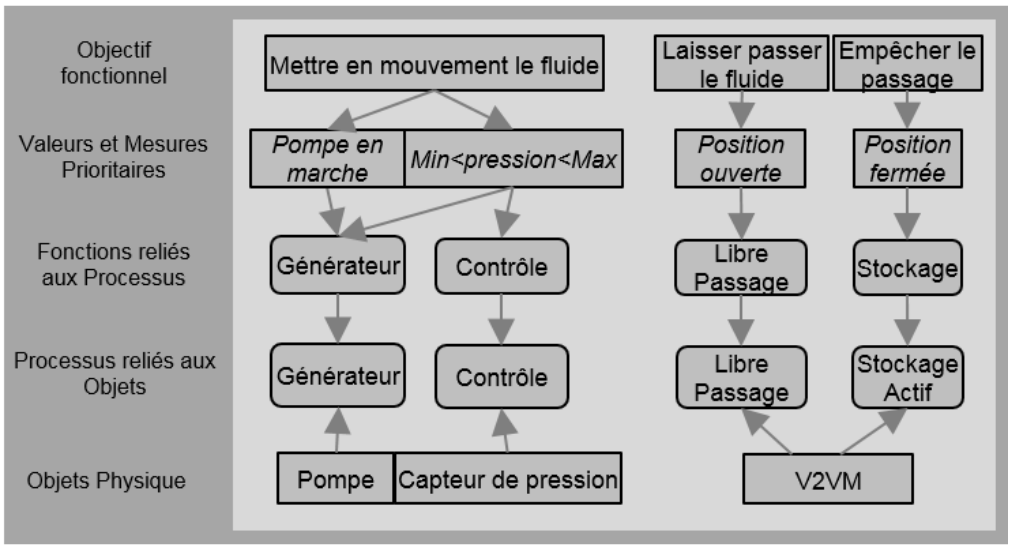


FIGURE 4.4 – Exemple de modélisation au niveau Composant

La Figure 4.4 est un exemple de modélisation pour une vanne à deux voies motorisée (V2VM) et une pompe centrifuge. L'objectif fonctionnel de la vanne est de permettre le passage du fluide, ou au contraire d'empêcher le passage du fluide. L'atteinte de ces objectifs est associé à une position de la vanne (ouverte ou fermée). Les fonctions de services associées sont ainsi une fonction de *Libre Passage* et une fonction de *Stockage* qui sont respectivement implémentées par une opération de *Libre Passage par la vanne* et une opération de *Stockage*

Actif par la vanne. L'objectif fonctionnel de la pompe centrifuge est de mettre en mouvement le fluide en imposant une pression. Une fonction de *Générateur* doit ainsi être implémentée par une opération de *Générateur par la pompe*. Une fonction de *Contrôle* doit également être implémentée par une opération de *Contrôle par le capteur de pression*.

La modélisation à ce niveau de décomposition permet ainsi de connaître selon l'opération (niveau *Objets reliés aux Processus*) qui est réalisée par le composant ce que l'on doit observer (niveau *Valeurs et Mesures Prioritaires*).

4.3.1.2 Modélisation au niveau Fonction

Pour pouvoir étendre le cadre de modélisation aux autres niveaux de décomposition, d'autres fonctions doivent être définies. Deux fonctions contextuelles sont introduites avec la particularité d'être mises en oeuvre par plusieurs opérations simultanément.

Définition 4.9 (fonction de Routage $X \rightarrow Y$)
Le but d'une fonction de **Routage** est d'aiguiller le fluide a travers la tuyauterie d'un point X vers un point Y, selon le contexte. Une fonction de *Routage* est mise en oeuvre par une ou plusieurs opérations de *Libre Passage par des vannes*, et potentiellement une ou plusieurs opérations de *Stockage Actif par des vannes*.

Définition 4.10 (fonction de Transfert $X \rightarrow Y$)
Le but d'une fonction de **Transfert** est de déplacer le fluide d'un point X vers un point Y. Une telle fonction requiert la mise en oeuvre d'une fonction de *Routage* $X \rightarrow Y$ et d'une fonction de *Générateur*.

Parmi les fonctions de haut niveau que doit réaliser le système, nous nous intéresserons en particulier à la fonction de *Transfert entre deux soutes* (Figure 4.5).

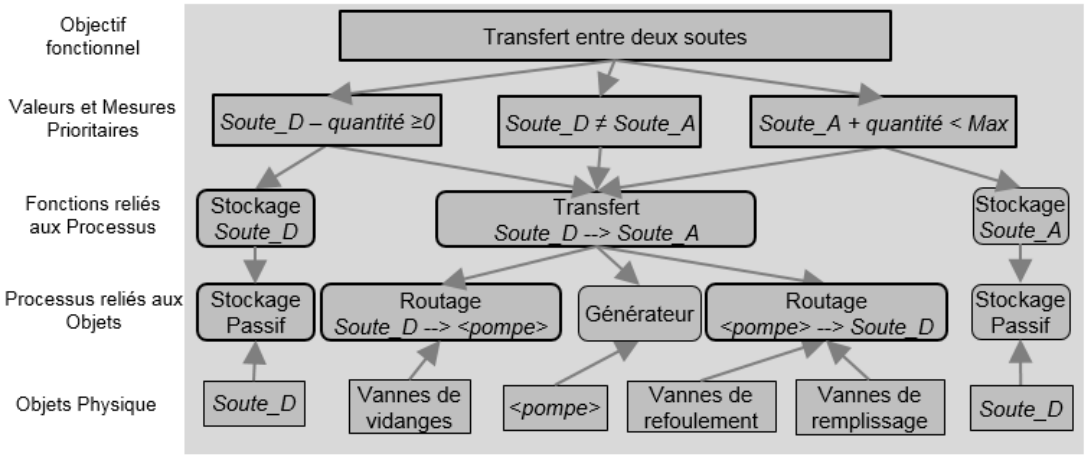


FIGURE 4.5 – Exemple de modélisation niveau Fonction

Pour réaliser un transfert entre deux soutes, la quantité à transférer doit être initialement présente dans la soute de départ et la soute d'arrivée doit avoir assez d'espace libre pour pouvoir contenir la quantité à transférer. De plus, une fonction de *Transfert* ainsi qu'une

fonction de *Stockage* doivent être mises en oeuvre. Les vannes de vidange de la soute de départ (*Soute_D* sur la Figure 4.5 implémentent la fonction de *Routage* de la soute vers la pompe qui réalise l'opération de *Générateur*. Les vannes de refoulement (de la pompe) et les vannes de remplissage mettent en oeuvre la fonction de *Routage* entre la pompe et la soute d'arrivée (*Soute_A*). Enfin, la soute d'arrivée réalise une opération de *Stockage Passif*.

Dans cet exemple, la représentation d'une configuration n'est pas assez précise. Il s'avère que selon l'architecture du système et le contexte intentionnel, une fonction de *Routage* peut être implémentée de différentes manières en utilisant les positions des vannes. Ainsi, il peut être très utile de définir des sous types pour la fonction de *Routage*, afin de prendre en compte le contexte. Pour clarifier notre propos, nous avons choisi d'illustrer nos remarques sur une instance spécifique d'une fonction de *Transfert entre deux soutes* (cf Figure 4.6).

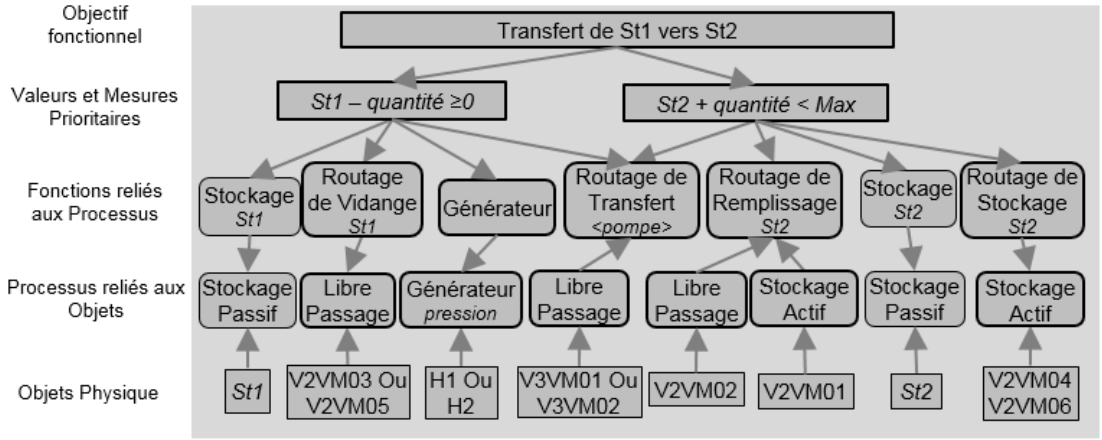


FIGURE 4.6 – Exemple de modélisation pour un Transfert entre St1 et St2

La fonction *Routage de Vidange* est mise en oeuvre par une opération de *Libre Passage* réalisée par une des vannes de vidange de la soute St1, selon quelle est la pompe qui implémente la fonction de *Générateur*. Dans notre cas d'étude, il s'avère que les vannes de vidange (des soutes) sont aussi les vannes d'aspiration (des pompes). Ce qui signifie que, si la pompe H1 est utilisée, alors une opération de *Libre Passage* est réalisée par la vanne V2VM03. De la même façon, la réalisation d'une opération de *Libre Passage* par une vanne de refoulement dépend de la pompe qui est utilisée.

En conséquence, l'ensemble des configurations possibles est plus explicite grâce à la contextualisation. Néanmoins, la distinction entre configuration minimale et maximale est implicite.

4.3.1.3 Modélisation au niveau Système

Au niveau Système, il peut être utile de représenter les configurations minimales et maximales pour un objectif donné (ou une fonction de service donnée). La Figure 4.7 est une représentation partielle de l'application du cadre de modélisation au système considéré.

Seuls deux objectifs fonctionnels sont représentés, à savoir : le maintient de la disponibilité en eau, et la distribution d'eau à l'équipage. Pour atteindre ces deux objectifs un critère de

potabilité doit être rempli. Pour maintenir la disponibilité en eau, le stockage de l'eau doit être maintenu au dessus du niveau minimum et en dessous du niveau maximum. De plus, pour pouvoir maintenir la disponibilité en eau, un approvisionnement en eau douce est nécessaire. Quatre fonctions de service génériques ont ainsi été identifiées :

- Production d'eau douce à partir d'eau de mer – mise en oeuvre par deux fonctions techniques (une fonction de Transformation et une fonction de Routage de Production) ;
- Stockage d'eau – mise en oeuvre par une opération de Stockage Passif et potentiellement par des opérations de Stockage Actifs réalisées par les vannes de vidange ;
- Transfert de l'eau d'un point à un autre – mise en oeuvre par au moins une fonction de Routage et une fonction de Générateur ;
- et Traitement de l'eau.

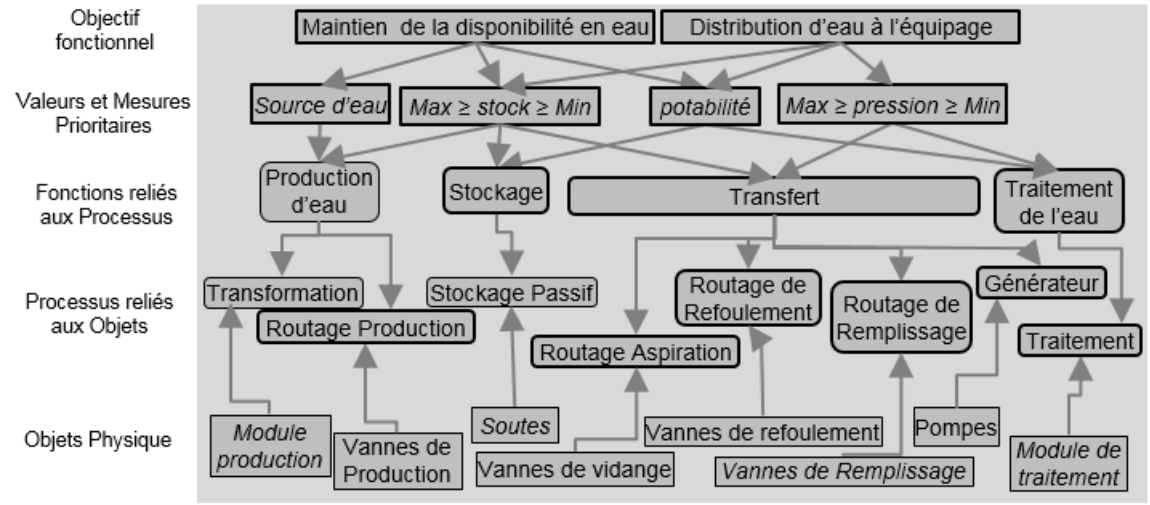


FIGURE 4.7 – Exemple partiel de modélisation au niveau Système

Ainsi, selon les objectifs fonctionnels qui doivent être atteints (définis par le scénario de test), on peut savoir quelles fonctions doivent être implémentées. Pour connaître la configuration du système, à partir des fonctions requises pour atteindre les objectifs, on se reporte à la modélisation au niveau Fonction pour connaître les configurations de ces fonctions. Par exemple, s'il est nécessaire de mettre en oeuvre une fonction de production d'eau pour pouvoir maintenir la disponibilité en eau, la configuration du système sera constituée d'une configuration de la fonction Transformation et d'une configuration de la fonction Routage de Production. L'ensemble des possibilités pour ces configurations sera donné par la modélisation au niveau Fonction.

Les aspects de reconfiguration sont maintenant pris en compte. Lorsque la situation évolue et qu'elle entraîne un changement d'objectifs à atteindre, on peut déterminer quel est le changement de configuration du système. Par exemple, le passage d'une configuration de Production d'eau vers une configuration de Transfert. Au contraire, lorsque le système doit poursuivre un même objectif malgré les évolutions (changement de contexte ou aléas), la configuration du système doit rester une configuration des fonctions qui doivent être implémentées.

Par exemple, si la configuration du système était une configuration de Production d'eau, une configuration de Production d'eau doit toujours être mise en oeuvre.

L'ensemble des configurations (possibles) peut ainsi être modélisé à différents niveaux de décomposition. D'autre part, cette modélisation peut aussi être utilisée pour l'analyse des résultats de simulation, pour savoir si le système de conduite est capable d'emmener le procédé dans une configuration qui délivre les services requis à un instant donné. Ceci fera l'objet de la section suivante.

4.3.2 Une méthodologie pour analyser les résultats de simulation

A partir de la modélisation multi-niveau et du scénario à exécuter, le comportement attendu du procédé peut être facilement déduit, en termes de configuration à des instants donnés. Puis, une fois le scénario exécuté, les résultats peuvent être reportés sur la modélisation pour être comparés aux configurations attendues. Pour illustrer les principes de nos propos, nous allons nous appuyer sur le cas d'étude représenté sur la Figure 4.8. Ce système est composé de deux soutes (Tank1 et Tank2), de deux pompes centrifuges (Pump1 et Pump2), et de six vannes à deux voies motorisées.

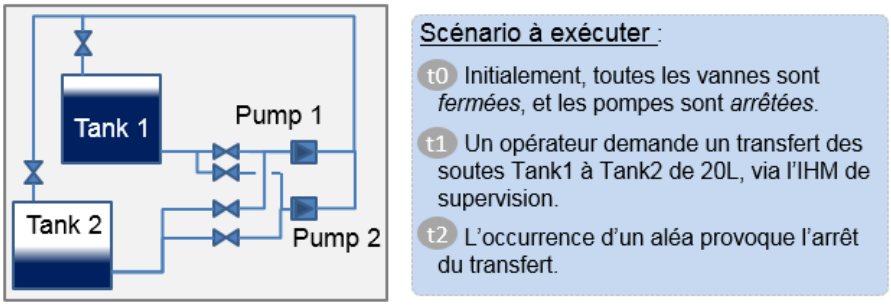


FIGURE 4.8 – Cas d'étude

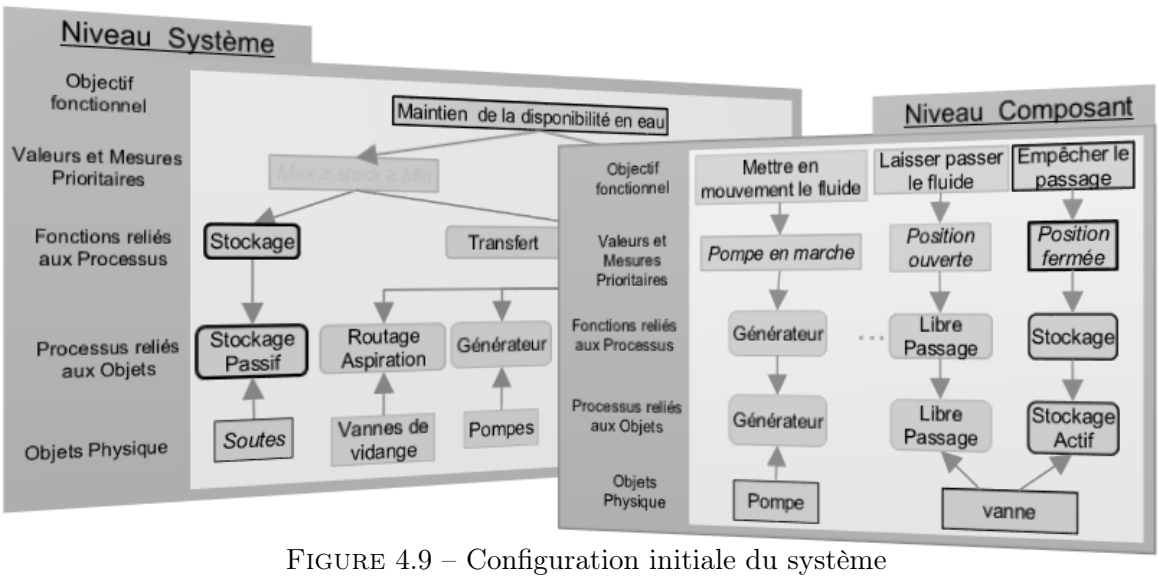


FIGURE 4.9 – Configuration initiale du système

A partir de la modélisation des propriétés et du scénario, on peut en déduire l'état initial du système, à savoir une configuration de stockage pour les deux soutes (cf Figure 4.9). De même, on peut définir les configurations attendues du système entre t1 et t2, puis après t2. Ainsi, suite à la demande de l'opérateur, le système devrait se trouver dans une configuration de Transfert entre les soutes Tank1 et Tank2. Suite à l'occurrence de l'aléa, une reconfiguration devrait avoir lieu pour permettre la poursuite de Transfert, le système devrait se retrouver dans une autre configuration de Transfert entre les soutes Tank1 et Tank2.

Passons maintenant à l'étape d'analyse des résultats de la simulation, suite à l'exécution du scénario. Après l'instant t1 où l'opérateur demande le transfert, les résultats de la simulation (Figure 4.10) indiquent que la pompe Pump1 est *en marche*, que la vanne de remplissage de la soute Tank2 est *ouverte*, que la vanne de vidange de la soute Tank1 reliée à la pompe Pump1 est *ouverte* et que toutes les autres vannes sont *fermées*.

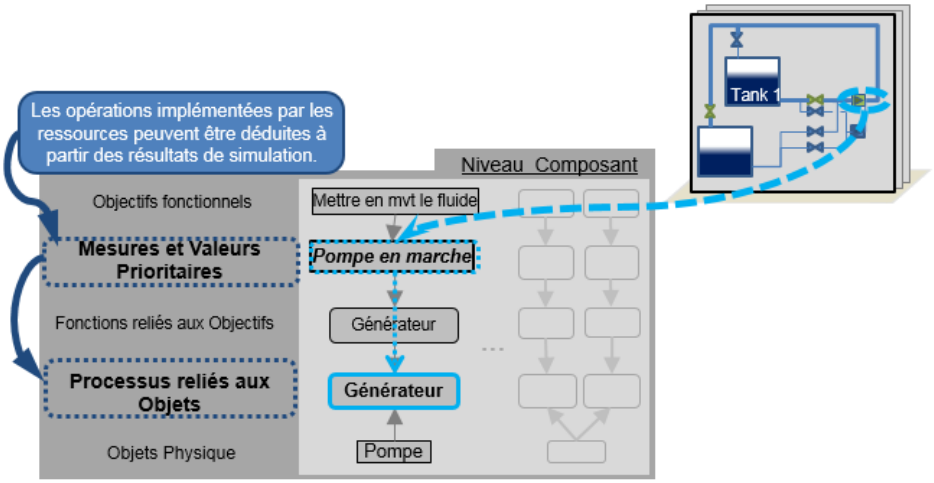


FIGURE 4.10 – Première étape pour analyser les résultats de la simulation d'un instant donné

Pour pouvoir comparer l'état du système (les résultats de la simulation) avec la configuration attendue, il nous faut savoir quelles opérations sont implémentées. Pour ce faire, on utilise la modélisation au niveau Composant, comme illustré sur la Figure 4.10. En projetant l'état du composant sur sa représentation dans le modèle au niveau d'abstraction Valeur et Mesures Prioritaires, on en déduit quelles opérations sont implémentées à cet instant donné. Ces informations peuvent ensuite être reportées sur la modélisation de niveau Fonction, afin de savoir quelles fonctions sont mises en oeuvre (cf Figure 4.11). Ainsi, au niveau Fonction, une fois que toutes les opérations réalisées ont été reportées, on peut savoir, par une lecture ascendante, à quelle(s) configuration(s) de fonction(s) correspond la configuration du système. Comme illustré sur la Figure 4.11, on peut voir que la configuration du système correspond à une configuration de la fonction de Transfert entre Tank1 et Tank2. La configuration du système correspond donc bien à ce que l'on s'attendait à avoir. En reportant ces informations au niveau Système, on pourra constater qu'une reconfiguration a eu lieu, puisque l'on est passé d'une configuration de Stockage à une configuration de Transfert (cf Figure 4.12). D'autre part, on peut dire que la configuration du système est correcte par rapport aux objectifs du système et à la situation.

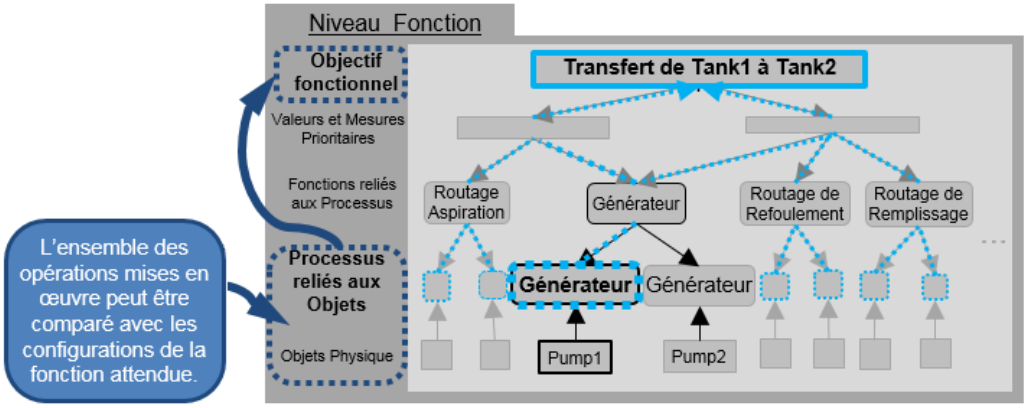


FIGURE 4.11 – Deuxième étape pour analyser les résultats de simulation d'un instant donné

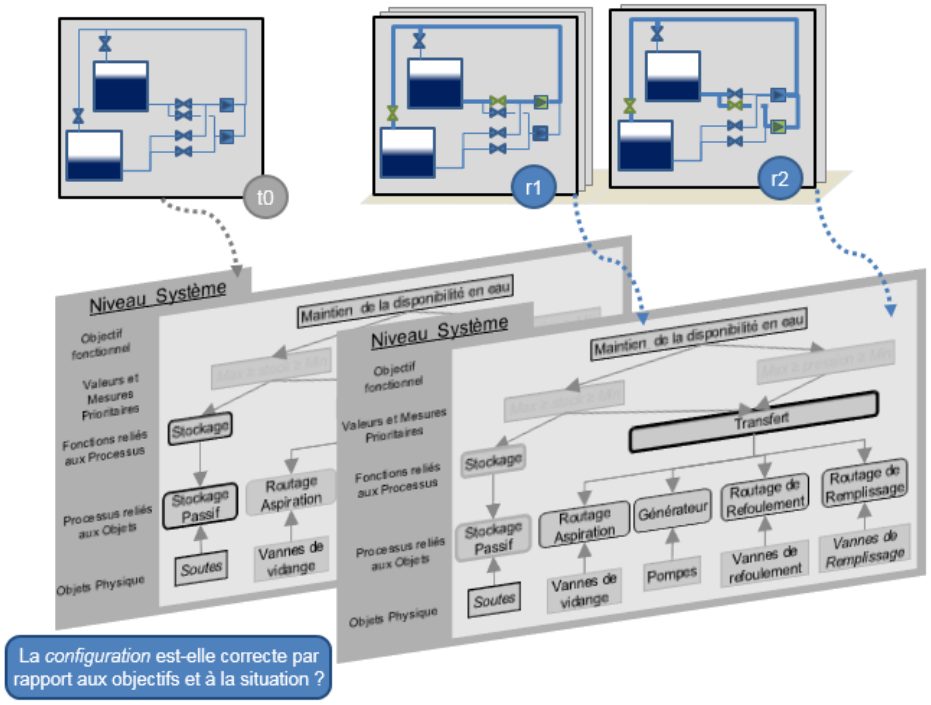


FIGURE 4.12 – Analyse des résultats de la simulation au niveau Système

Suite à l'occurrence de l'aléa (défaillance de la pompe Pump1), la configuration de Transfert qui était implémentée ne permet plus de réaliser la fonction (puisque cela nécessitait la réalisation d'une opération de Générateur par cette pompe). Les résultats de simulation, suite à cette panne, indiquent que la pompe 2 est en marche (cf Figure 4.12). En appliquant la démarche pour analyser les résultats, on s'aperçoit que le système est toujours dans une configuration de Transfert de Tank1 à Tank2. Toutefois ce n'est pas la même puisque c'est la pompe Pump2 qui réalise l'opération de Générateur. Ainsi, on constate qu'il y a eu une reconfiguration permettant au système de poursuivre sa mission, le système est passé d'une configuration de fonction de Transfert à une autre.

4.4 Bilan/Synthèse

Afin de faciliter la mise en oeuvre de tests tout au long de la conception, nous avons montré que les propriétés à vérifier doivent être formalisées dans le contexte. S'agissant de s'assurer qu'un système de conduite est capable d'emmener le procédé dans un état permettant de fournir les services requis, l'objectif de notre contribution vise à faciliter l'interprétation des résultats de la simulation par cette formalisation.

A cet égard, nous proposons un formalisme de modélisation multi-niveau des propriétés (Figure 4.13), en utilisant la Hierarchie d'Abstraction de l'analyse du domaine de travail (WDA), qui peut être vue comme une analyse fonctionnelle, avec les notions d'*opération* et *configuration* issues des systèmes de production flexible. La Hiérarchie d'Abstraction nous permet de représenter le contexte intentionel (les objectifs du système avec les critères de performance) reliés aux moyens pour atteindre les objectifs (quelle(s) fonction(s) doit être implémentée(s) et quelles ressources peuvent le faire). La Hiérarchie d'Abstraction n'étant qu'un outil de modélisation, nous avons utilisé le concept d'*opération* (fonction technique réalisée par une ressource) pour formaliser les informations. L'état du système est quant à lui exprimé en terme de *configuration* (l'ensemble des opérations réalisées par les ressources). Ainsi, le comportement attendu, correspondant à un ensemble d'états du procédé à des instants donnés, peut être exprimé dans le contexte.

D'autre part, afin de réduire l'écart entre *comportement attendu* et *comportement obtenu*, nous proposons à partir de la modélisation des propriétés une méthodologie pour interpréter les résultats de la simulation. A partir du scénario à exécuter, le comportement attendu est dérivé de la modélisation des propriétés, à des instants donnés, définissant ainsi les configurations attendues. Puis, suite à l'exécution du scénario, les résultats de la simulation, à un instant donné, sont projetés sur la modélisation multi-niveau (via une approche ascendante) afin d'être comparés avec la configuration attendue (Figure 4.13). Ainsi le comportement attendu (ensemble des configurations attendues à des instants donnés) est facilement comparé avec le comportement obtenu (ensemble des configurations obtenues à des instants données).

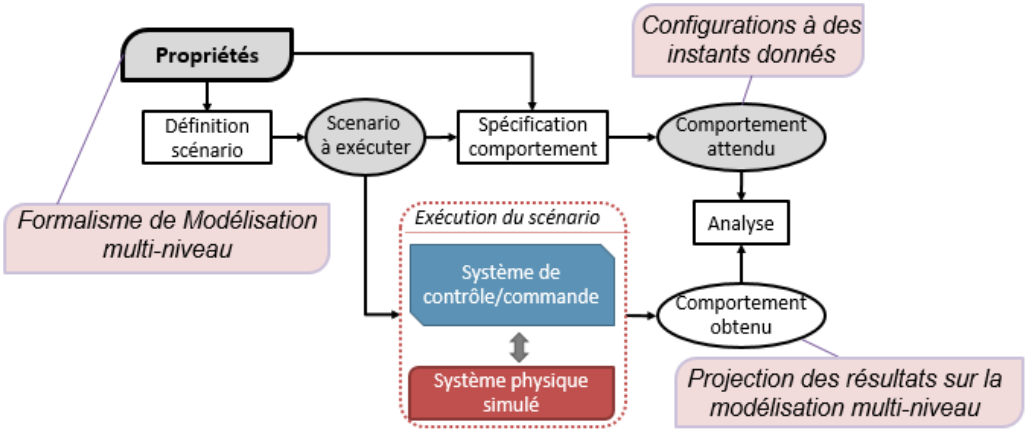


FIGURE 4.13 – Les apports de notre proposition pour analyser les résultats de la simulation

Nous avons appliqué la proposition de modélisation des propriétés, à différents niveaux de

décomposition (composants, fonctions de haut niveau et système), d'un système de gestion de fluide. Au travers de cette modélisation, le caractère reconfigurable a pu être testé à différents niveaux de granularité : changement de configuration au niveau système (changement d'objectif à atteindre) et changement de configuration d'une fonction de haut niveau (suite à une défaillance d'un élément du procédé). Pour interpréter les résultats issus de la simulation, nous avons utilisé notre proposition de méthodologie, permettant ainsi de réduire les écarts de représentation entre comportement attendu et comportement obtenu. Notre contribution apporte ainsi, outre une formalisation des propriétés contextualisées, une méthodologie pour faciliter l'interprétation des résultats de la simulation (Figure 4.13) dont la validité a été démontrée sur un exemple applicatif.

Notre proposition offre l'avantage de facilement pouvoir dériver le comportement attendu par rapport au scénario à exécuter, contrairement à une modélisation SysML qui demande une étape de synthèse d'informations. Elle offre d'autre part l'avantage de pouvoir facilement comparer les comportements attendu et obtenu. Au travers de la méthodologie présentée, la vérification du caractère adaptable du système de conduite s'en trouve facilitée. Les propriétés sont formalisées dans le contexte et de façon non ambigüe, facilitant ainsi l'interprétation des résultats de la simulation. De plus, le formalisme de modélisation proposé permet de tester un système de conduite dans son ensemble ou bien seulement une partie (par exemple : la partie de contrôle-commande ou la partie de supervision). De ce fait, la méthodologie proposée peut également être utilisée lors de tests utilisateurs (tests de validation). Toutefois, le formalisme proposé ne permet pas la représentation des priorités entre les fonctions, qui doivent donc être exprimées à part. Pour spécifier les fonctions qui peuvent ou non être mises en oeuvre en même temps selon le contexte (contraintes de sécurité ou de fonctionnement), des outils de modélisation utilisés dans la phase suivante du CWA pourraient être utilisés.

Dans ce chapitre, nous avons levé ce premier verrou au travers d'une proposition de formalisme de modélisation multi-niveaux des propriétés, qui a fait l'objet d'une publication [Prat et al., 2016]. Nous avons également apporté une première contribution pour faciliter la mise en oeuvre de tests, en proposant une méthodologie qui facilite l'interprétation des résultats de simulation. Dans les prochains chapitres, nous chercherons à faciliter l'obtention des modèles de simulation, afin de pouvoir intégrer la mise en oeuvre de tests tout au long de la conception.

5

Structuration des modèles de simulation

Sommaire

5.1 Introduction

L'utilisation de modèle de simulation du procédé pour tester le système de conduite a un double intérêt. Outre la possibilité de réaliser des tests tôt dans la conception, des situations anormales de fonctionnement, potentiellement dangereuses, peuvent être artificiellement reproduites, sans engendrer de dégâts matériels ni porter atteinte à la sécurité des êtres humains. Néanmoins, l'utilisation de la simulation comme technique de vérification permet uniquement de conclure sur les cas traités. Par conséquent, de nombreux tests doivent être réalisés pour couvrir un maximum de cas.

Il apparait donc une contradiction : d'un côté il est important de réaliser autant de tests que possible et au plus tôt (ce qui augmente les délais de conception), d'un autre côté pour rester compétitif les délais de conception tendent à être minimisés. Or, l'obtention de la simulation est de façon générale une tâche coûteuse (en temps et en argent) [Robinson, 2004]. Contradiction qui est amplifiée si l'on considère la mise en oeuvre de tests en parallèle des activités de conception, augmentant non seulement les délais de conception mais aussi les coûts. En effet, les modèles de simulation sont conçus dans une intention particulière, or, la nature des vérifications entre le début et la fin du projet sont très différentes (qualitatif vs quantitatif). En conséquence, il y a un fossé entre ces objectifs de simulation, en termes de

modélisation et de simulation. Tandis qu'en début de projet le comportement du procédé peut être vu comme un système à événement discret, plus le projet avance plus le niveau de fidélité dans la reproduction du comportement du procédé augmente. L'introduction de vérifications par simulation tout au long de la conception requiert ainsi le développement de multiples modèles de simulation et nécessite potentiellement différents simulateurs (logiciels de simulation).

Bien que le développement d'une simulation ne soit plus essentiellement une activité de programmation, l'obtention manuelle des modèles de simulation est longue et potentiellement fastidieuse. Un intérêt, dans la littérature, est porté sur la génération automatisée de ces modèles, outillant la phase d'implémentation dans le processus de développement d'une simulation. Cependant, un effort important de modélisation est toujours nécessaire. Un savoir faire en simulation est également requis, ne serait-ce que pour les choix technologiques concernant l'implémentation. Ainsi, le déploiement dans un contexte industriel d'approches de type *virtual commissioning*, bien que prometteuses, doit faire face :

- à un important effort de modélisation [Lee and Park, 2014], [Dahl et al., 2016],
- à un savoir faire en simulation [Drath et al., 2008], [Oppelt and Urbas, 2014]
- au maintien à jour des modèles de simulation [Barth and Fay, 2013].

La simulation étant une activité basée sur les modèles, un intérêt dans la littérature est porté sur la *réutilisation*, afin de réduire les coûts. Il s'agit, par exemple, de réutiliser des données d'ingénierie pour générer (automatiquement) les modèles de simulation, et/ou de réutiliser des modèles de simulation. On peut citer en exemple des travaux dans le cadre du *virtual commissioning* tel que [Barth and Fay, 2013], [Hoffmann et al., 2010], [Oppelt et al., 2014]. En intégrant le *virtual commissioning* en parallèle des activités de conception, l'effort de modélisation peut être atténué, selon [Oppelt and Urbas, 2014], via la réutilisation des précédents modèles de simulation. L'idée est de commencer par un modèle de simulation "simple" (pour des tests en début de conception) qui est ensuite enrichi au fur et à mesure que le projet avance. Ainsi, l'effort de modélisation serait réparti sur la durée des activités de conception [Oppelt and Urbas, 2014]. Toutefois, de par la nature hétérogène du procédé à simuler, et l'intention particulière dans laquelle sont construits les modèles, il n'est pas toujours possible ou souhaitable de réutiliser les modèles de simulation, l'effort pour adapter les modèles pouvant être très important et donc contre-productif [Robinson, 2004].

Faciliter la mise en oeuvre de tests en parallèle de l'avancement de la conception, passe ainsi par faciliter la construction des modèles de simulation. Pour réduire les coûts et les délais, un intérêt est porté sur la réutilisation des modèles de simulation et sur l'automatisation de la génération des modèles de simulation. Cependant les objectifs d'utilisation de la simulation dépendent des vérifications que l'on souhaite effectuer, et de l'instant où elles sont effectuées dans le processus de conception. En effet, tester la partie commande nécessite de spécifier le comportement du système physique aux niveaux des entrées/sorties, tandis que le test du système de conduite dans son ensemble, nécessite de représenter toute la chaîne fonctionnelle (des capteurs/actionneurs jusqu'au comportement mécanique/physique) [Hoffmann et al., 2010].

Il faut donc afin de pouvoir faciliter la construction des modèles de simulation, définir une

structuration générique des modèles du procédé, qui soit adaptable aux différents besoins de modélisation. Ceci correspond au verrou 2 identifié dans le Chapitre 3 « *comment structurer les modèles de simulation pour permettre la réutilisation* ». Afin de lever ce verrou, un état de l'art est réalisé dans la section qui suit. Puis, section 5.3, nous proposons une structuration des modèles de simulation.

5.2 Etat de l'art

De façon générale, le comportement d'un système peut être décrit à différents niveaux d'abstraction ou détail ainsi que par différents formalismes [Vangheluwe et al., 2002]. Comme évoqué dans la littérature, différents niveaux d'abstraction sont souvent nécessaires, le choix d'un niveau de détail adéquat selon les objectifs de la simulation est un point important [Benjamin et al., 1998]. Néanmoins, le choix d'un niveau d'abstraction et d'un formalisme dépend du système à représenter mais aussi du contexte, de l'expérience et des objectifs du modélisateur [Vangheluwe et al., 2002].

Dans le cadre du *virtual commissioning*, les modèles de simulation sont principalement constitués par décomposition granulaire d'éléments de simulation ou objets, qui représentent plusieurs parties, composants ou modules du système considéré [Puntel-Schmidt and Fay, 2015]. Les modèles de simulation sont généralement des modèles mixtes continus/discrets, pour représenter le comportement physique du procédé (de l'usine) et des capteurs et actionneurs [Puntel Schmidt and Fay, 2015]. Partant du constat que peu d'approches existent pour définir le type de formalisme et le niveau de détail à utiliser, un aperçu de types de modèles selon le niveau de détail est proposé dans [Puntel-Schmidt and Fay, 2015]. Ceci illustre bien le besoin de faciliter l'obtention de modèles de simulation afin de pouvoir effectuer des tests tout au long de la conception.

En vue de proposer une approche de génération des modèles de simulation qui puisse être utilisée tout au long de la conception d'un système de conduite de procédé, une stratégie de réutilisation doit être mise en oeuvre. Au travers de la revue de la littérature, on s'intéressera dans un premier temps à la notion de *réutilisation*. Puis, nous nous intéresserons à celle-ci dans le contexte de *génération automatisée de modèles de simulation*. Ceci permettra de faire émerger les pistes de recherches pour faciliter l'obtention de simulations en parallèle de l'avancement de la conception.

5.2.1 Réutilisation et composition de modèles : les principes

Le concept sous-jacent de la réutilisation, selon [Reese and Wyatt, 1987], est l'identification et l'extraction de composants logiciels à partir d'autres projets. Un composant correspond à n'importe quel élément d'un système logiciel, et peut inclure des éléments provenant de la spécification des exigences, de la conception, de l'implémentation et du test. Tandis qu'un composant logiciel fait référence à tout aspect documentable du projet (par exemple guide utilisateur, diagramme de flux, plan de test) [Reese and Wyatt, 1987]. La mise en place d'une

stratégie de réutilisation comprend, outre l'utilisation des composants, la sélection, l'incorporation, la maintenance et la gestion des composants en général [Reese and Wyatt, 1987]. Cela passe ainsi par la mise en oeuvre de mécanismes pour faciliter la localisation/comparaison/sélection d'artefacts (à réutiliser), de mécanismes pour combiner, connecter et faire communiquer les artefacts réutilisés [Pidd, 2002].

Réutiliser des modèles de simulation peut avoir plusieurs significations : cela peut aller de la réutilisation de portions de code, à la réutilisation de modèles complets, en passant par la réutilisation de composants [Robinson et al., 2004]. Toutefois, un modèle étant conçu dans une intention particulière, avant de pouvoir le réutiliser il faut d'abord évaluer s'il convient au besoin. Il peut donc s'avérer difficile de réutiliser un modèle complet dans un autre contexte que celui pour lequel il a été prévu. Cela pose le problème de la validité du modèle pour la nouvelle utilisation ainsi que de sa crédibilité [Pidd, 2002]. Ainsi, il peut être plus rapide de développer son propre modèle plutôt que de chercher à réutiliser des modèles existants, d'autant plus qu'il est souvent nécessaire de les adapter au nouveau besoin [Robinson et al., 2004]. Par conséquent, lorsque l'on conçoit un modèle, il faut anticiper le futur besoin d'utilisation (c'est à dire de réutilisation). Le modélisateur n'est pas uniquement dans une activité productive, il est aussi dans une activité constructive. Selon [Cetinkaya et al., 2010], l'utilisation d'une approche basée sur les composants dans le domaine de la simulation pourrait être bénéfique, via la réutilisation de composants interopérables et une modélisation hiérarchique. Pour faciliter la réutilisation, un paradigme orienté objet est généralement utilisé pour la modélisation et la programmation. Comme évoqué par Pidd, il est plus facile de réutiliser des portions de code, ou des fonctions que des modèles/programmes entiers [Pidd, 2002], [Robinson, 2004]. L'approche basée sur les composants (dans le génie logiciel) correspond à ce qui est appelé composabilité (de simulation) dans le domaine de la simulation [Bartholet et al., 2004].

La mise en place de stratégies permettant de composer un modèle (de simulation) à partir d'autres modèles (stockés dans une bibliothèque, par exemple) vise à réduire les coûts et les temps de développement de simulation. La composition de modèles a pour objectif d'assembler des composants (objets) de taille et de complexité arbitraire [Robinson, 2004]. L'intérêt de la composabilité est la flexibilité de créer des objets réutilisables à des niveaux de granularité variants [Robinson, 2004]. Néanmoins, malgré un désir de réutilisation de modèles "complets", ce n'est pas toujours évident. La composabilité peut tout aussi bien introduire de la complexité dans la modélisation, comme la réduire [Page and Opper, 1999]. Il est à noter, que *réutilisabilité* et *composabilité* sont deux choses sensiblement différentes, comme l'indique les définitions données dans [Balci et al., 2011]. La *composabilité* y est définie comme le degré avec lequel un artefact est capable d'être établi en combinant des choses, des parties, des éléments. La *réutilisabilité* y est quant à elle définie comme le degré avec lequel un artefact, une méthode ou une stratégie est capable d'être utilisée plusieurs fois ou de façon répétée. Les problématiques liées à la composition de modèles rejoignent ainsi celles liées à la combinaison de modèles (modélisation hétérogène) dans le cas de simulation combinée.

De façon générale, lors de la modélisation d'un système, on peut avoir des modèles homogènes (utilisation de formalismes et paradigmes identiques) ou bien des modèles hétérogènes

(paradigmes et/ou formalismes différents). Dans le cas de modèles hétérogènes, on peut définir trois situations possibles [Hardebolle, 2008], la dernière étant la plus problématique : différents formalismes sont utilisés mais avec un unique paradigme sous-jacent ; un unique formalisme est utilisé mais avec des paradigmes différents ; ou bien l'usage de formalismes et de paradigmes différents. Pour implémenter (et simuler) un modèle combiné (plusieurs formalismes), il existe deux types d'approches : soit on utilise une approche de multi-modélisation (un seul simulateur), soit on utilise la co-simulation (autant de simulateurs que nécessaire). Lorsque le modèle couplé est constitué de sous-modèles exprimés dans des formalismes différents, on peut utiliser un super-formalisme ; transformer les sous-modèles dans un formalisme commun ou bien utiliser une approche de co-simulation [Vangheluwe et al., 2002]. La réutilisation, pour le développement d'une simulation, peut ainsi se faire à différents niveaux et au travers de différentes stratégies. Il peut s'agir de réutilisation pour construire le modèle de conception de la simulation, ou bien de la réutilisation de code pour implémenter le modèle de simulation (exécutable). Toutefois, la conception d'un modèle par composition (combinaison d'artefacts), peut introduire une certaine complexité, que ce soit pour la de modélisation en elle-même ou l'implémentation du modèle de simulation.

Intégrer des tests en parallèle des activités de conception de systèmes de conduite de procédés, nécessite de faciliter l'obtention de la simulation. Afin de réduire l'effort de modélisation qui est bloquant dans un contexte industriel, il a été suggéré, en particulier par [Oppelt and Urbas, 2014], d'enrichir les modèles de simulation au fur et à mesure de l'avancement des activités de conception. Néanmoins, il peut être difficile de réutiliser des modèles "complets" de composants, notamment si l'intention particulière pour laquelle ils ont été créés est différente. De plus, des difficultés peuvent apparaître si l'on cherche à réutiliser des modèles dont les méthodes de modélisation (pour la simulation) sont différentes. Réduire l'effort de modélisation requiert ainsi la mise en place d'une stratégie de réutilisation. Il s'agit notamment de prévoir :

- un mécanisme pour faciliter la localisation et la sélection des artefacts à réutiliser (via un bibliothèque artefacts de simulation, par exemple),
- un mécanisme pour pouvoir adapter et enrichir les modèles (de simulation) au fur et à mesure de l'avancement du projet, les besoins de modélisation évoluant.

Pour réduire le temps lié à l'obtention des modèles de simulation et réellement faciliter l'obtention de la simulation, la section suivante s'intéressera à la génération automatisée de simulation.

5.2.2 Réutilisation et composition : la génération automatique

Les logiciels de simulation bien que fournissant un environnement de modélisation qui dispose de librairies d'artefacts de simulation pour faciliter la construction du modèle de simulation, un important effort de modélisation doit tout de même être fourni. Différents travaux ont portés sur la génération automatisée des modèles de simulation, évitant ainsi de devoir saisir manuellement le modèle de simulation dans le logiciel (commercial) de simulation. Généralement, cette génération passe par l'utilisation d'une bibliothèque de modèles de

simulation *élémentaires* [Adam et al., 2012], qui sont ensuite instanciés et paramétrés, par rapport à un modèle de connaissance. Un des avantages, mis à part que la génération s'en trouve facilitée, est que cela permet une capitalisation de la connaissance et la réutilisation de modèles (et non plus de composants).

Par exemple, dans les travaux de [El Haouzi et al., 2007], l'implémentation d'une plate-forme de simulation générique de systèmes de production organisés en flux tirés est faite à partir d'un modèle de connaissance en UML et d'une bibliothèque de modèles génériques. Une instanciation de la plate-forme générique permet d'obtenir la plate-forme de simulation pour un système particulier, par un utilisateur non expert en modélisation. Pour modéliser le comportement de la partie physique du système, une approche orientée objet a été utilisée, permettant outre la cohérence entre toutes les étapes de construction des modèles de simulation, de pouvoir identifier chaque objet du système réel par un modèle. Parmi les logiciels permettant de simuler des flux par une approche orientée objet, leur choix s'est porté sur le logiciel ARENA (cas de la simulation à événement discret).

Dans un contexte de *virtual commissioning*, les travaux de [Barth and Fay, 2013] proposent d'utiliser des algorithmes de génération automatisée pour réduire l'effort de construction manuelle du modèle de simulation. La génération se fait en réutilisant les données d'ingénierie qui décrivent le procédé (schéma P&ID) et les librairies de simulation Modelica existantes. Les travaux de [Oppelt et al., 2014] proposent une approche propriétaire similaire de génération semi-automatisée, basée sur une suite logicielle Siemens. Les travaux de [Hoernicke et al., 2015] proposent une approche visant également à obtenir des modèles Modelica, mais dans le cadre de réhabilitation d'installations existantes. De ce fait, les données d'ingénierie si disponibles ne sont pas forcément à jour. L'interface de supervision est ainsi utilisée pour obtenir un modèle de référence. Ces travaux montrent qu'il est possible de générer le modèle de simulation du procédé en réutilisant les données d'ingénierie. Ainsi, le schéma P&ID, qui est un modèle métier, constitue un modèle de connaissance (modèle conceptuel). Néanmoins, bien qu'une approche orientée objet soit utilisée, la génération du modèle de simulation repose sur des librairies de simulation pré-existantes (représentation fine du comportement du procédé). Ces librairies ne sont donc pas adaptées pour des vérifications du système de conduite en début de conception.

Dans les travaux de [Foures et al., 2012], le modèle de connaissance est sous la forme d'un modèle SysML, où le diagramme d'activité est transformé en Réseau de Petri, puis transformé en VHDL-AMS pour être exécuté sur le logiciel System Vision. Les transformations sont effectuées en utilisant les techniques de l'Ingénierie Dirigée par les Modèles (IDM).

Les techniques de l'IDM sont également utilisées dans les travaux de [Lallican, 2007] concernant la conception de systèmes transitiques, et étendus par la suite aux systèmes transitiques reconfigurables par [Bévan, 2013]. Ces travaux traitent de la génération conjointe de code de commande et de modèles de simulation. La génération se fait à partir d'une description du système, exprimée dans un langage spécifique au domaine (DSL) et d'une bibliothèque de composants (au sens de Lallican). Cette bibliothèque est basée sur le concept de *vue*. Les vues *parties opératives* des composants sont utilisées pour obtenir, par transformation de modèle, le

modèle de simulation de la partie opérative du système transitique. Ce dernier est exécutable sur le simulateur SimSED. De plus, une approche de type Software-in-the-Loop permet de vérifier le code de commande généré. Dans le cadre de simulations en ligne, afin de pouvoir initialiser le modèle de simulation du système de production, les travaux de [Adam, 2013] permettent de générer conjointement la commande et l'observateur, à partir d'un même modèle de connaissance. Cet observateur, basé sur un simulateur à événement discret, a pour but de fournir au système d'Aide à la Décision l'état du système de production.

L'utilisation d'une bibliothèque basée sur le concept de *vue* est intéressant. Un même élément du procédé pourrait disposer de différents modèles de simulation contenus dans différentes *vues* de simulation, selon la vérification à effectuer et l'avancement du projet. D'autre part, l'utilisation d'une bibliothèque d'éléments, basée sur le concept de *vue*, permettrait d'envisager l'intégration d'une démarche de génération automatisée au sein d'un flot de conception automatisé de contrôle-commande. De façon générale, dans les méthodes de modélisation pour la simulation basée sur l'IDM, deux approches sont distinguées : l'une basée sur un langage unifié, l'autre sur un langage spécifique au domaine [Li et al., 2013]. L'approche *Model Driven Architecture* (MDA) consiste à transformer un modèle indépendant de toutes plateformes (PIM), défini dans un langage de modélisation unifié, en un modèle spécifique à une plateforme (PSM). Tandis que l'approche *Model Integrated Computing* (MIC) a pour but de fournir des solutions de modélisation spécifiques pour des domaines de problèmes. Ces approches visent ainsi une modélisation des modèles de simulation indépendamment de leurs implémentations, puis d'en générer le code exécutable (spécifique à une plateforme).

Afin, de générer les modèles de simulation en parallèle des activités de conception de système de conduite, l'utilisation de techniques basées sur l'IDM semble approprié. En effet, la conception du système de conduite et simulation sont toutes deux des activités à base de modèles. D'autre part, pour générer des modèles de simulation de procédé, il semble judicieux de réutiliser les modèles métiers utilisés pour la conception que ce soit du procédé ou de la partie commande (pour les échanges avec le procédé).

5.2.3 Bilan et pistes de recherches

Comme précédemment évoqué, nous visons la mise en place d'une stratégie de réutilisation ainsi qu'une approche de génération automatisée des modèles de simulation tout au long de l'avancement du projet. La mise en place d'une stratégie de *réutilisation* peut se faire à différents niveaux. Nous visons en particulier une réutilisation :

- des données d'ingénierie (modèles métiers décrivant le procédé à simuler) pour construire le modèle de simulation,
- de modèles de simulation élémentaires pour générer le modèle de simulation du procédé (via une bibliothèque de modèles de simulation),
- de l'approche de génération automatisée, durant tout le projet.

Dans le cadre des activités de conception, le paradigme orienté objet est utilisé pour la modélisation du système. L'utilisation de ce paradigme pour la phase de modélisation et d'implémentation est ainsi en adéquation avec nos objectifs de réutilisation (réutilisation de modèle

métier tel que schéma P&ID, et réutilisation de modèles de simulation d'éléments du procédé). Dans ce contexte de *conception orientée objet*, on peut ainsi envisager une approche de génération de modèles de simulation à partir d'une *bibliothèque de modèles de simulation* des éléments.

On voit ainsi se dessiner une esquisse de mécanisme pour localiser et sélectionner les artefacts de simulation à réutiliser, nécessaire pour la mise en place d'une stratégie de réutilisation. S'agissant de simuler le comportement des éléments du procédé présents sur le schéma P&ID, il est envisageable d'utiliser le concept de *vue* pour faire le lien entre la représentation de l'élément sur le schéma et son modèle de simulation. De plus, ce concept de *vue* pourrait également être utilisé pour disposer d'autant de modèles de simulation d'un élément que nécessaire. Ceci pourrait permettre de mettre en place une approche de génération du modèle de simulation du procédé (cf Figure 5.1) dans un flot de conception automatisée, tel que celui d'Anaxagore.

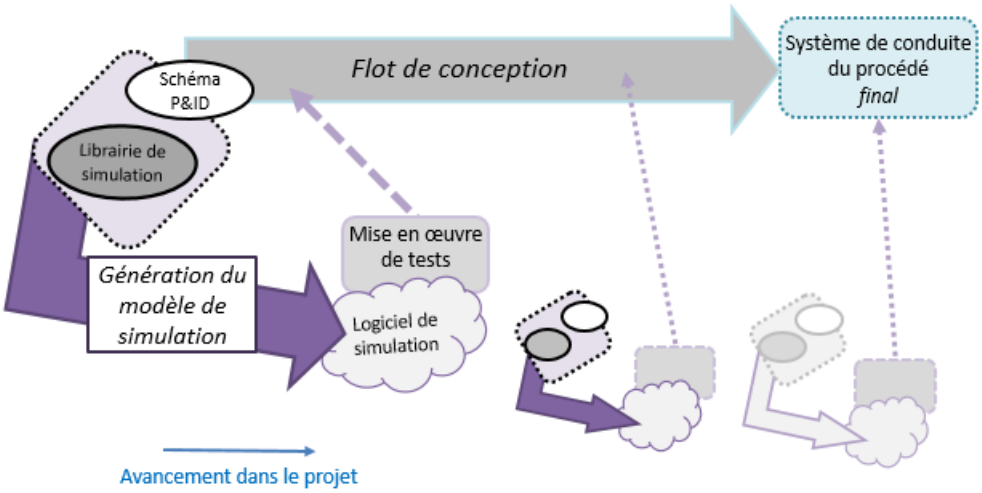


FIGURE 5.1 – Approche de génération envisagée pour faciliter les tests

D'autre part, afin de pouvoir mettre en oeuvre une approche de génération qui puisse être utilisée pendant toute la durée du projet, l'utilisation d'un langage de simulation orienté-objet et permettant une modélisation multi-domaine, tel Modelica, semble prometteur. Néanmoins, cela demande de développer des librairies de modèles de simulation, les librairies existantes n'étant pas adaptées pour des vérifications du système de conduite en début de conception. Afin de faciliter la construction des modèles de simulation, les modèles doivent être structurés pour pouvoir permettre la *réutilisation*, en tenant compte de l'évolution des besoins en termes de modélisation du comportement. Pour ce faire, nous proposons une structuration des modèles de simulation des éléments, objet de la suite du chapitre.

5.3 Proposition de Structuration des modèles de simulation

Comme évoqué dans l'état de l'art, une approche de modélisation modulaire permet notamment de faciliter les modifications, et par conséquent la réutilisation. Compte tenu du

comportement du procédé à dominante continue, en particulier celui du fluide, et celui de la partie opérative plutôt à événements discrets, un intérêt est porté sur une structuration des modèles des éléments permettant de dissocier ces deux types de comportement.

Dans un premier temps nous poserons les principes pour structurer les modèles de simulation des éléments du procédé. Puis, nous appliquerons ses principes sur un cas d'étude simplifié, pour obtenir le modèle de simulation dans le langage choisi, à savoir Modelica.

5.3.1 Principes de la proposition

Pour représenter le comportement du procédé nous proposons d'utiliser une approche de décomposition pour modéliser d'une part le comportement plutôt à événements discrets de la partie opérative (actionneur et l'instrumentation), et d'autre part du comportement du fluide à dominante continue. Cette approche permet notamment de dissocier la modélisation du comportement d'un élément de la partie opérative (qui n'est pas dépendant du fluide), de la modélisation des effets de la partie opérative sur le fluide (qui, elle, est dépendante du fluide). De plus, selon les vérifications à effectuer, les besoins en termes de modélisation du comportement du fluide et/ou de la partie opérative varient. L'utilisation du principe de décomposition a l'avantage de permettre de ne modifier qu'un sous-modèle du modèle d'un élément (plutôt que de devoir refaire toute la modélisation).

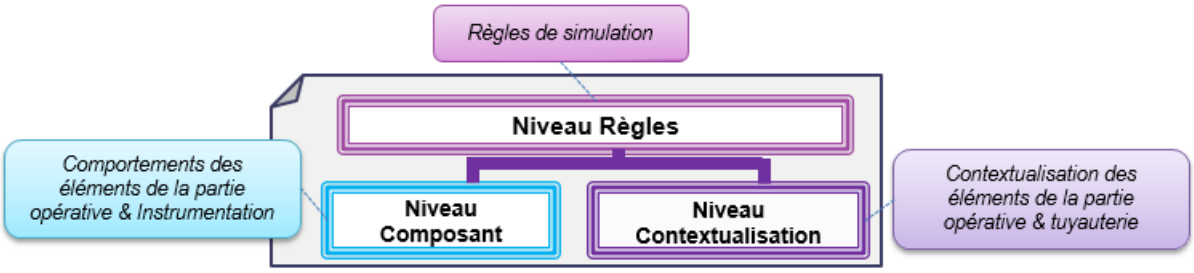


FIGURE 5.2 – Approche de structuration des modèles

Afin de représenter les éléments qui constituent un procédé, nous proposons donc d'utiliser le principe de décomposition, comme illustré sur la Figure 5.2. De façon générale, le modèle d'un élément est constitué d'une structure hiérarchisée, dans laquelle on distingue la modélisation du comportement générique de l'élément (*Niveau Composant*) et celle contextualisée (*Niveau Contextualisation*) qui permet de représenter les effets sur le fluide. Le *Niveau Règles*, étant le plus haut niveau d'abstraction, permet, entre autres, de faire le lien entre les sous-modèles (*Niveau Composant* et *Niveau Contextualisation*).

La Figure 5.3 montre la structure du modèle de simulation d'un élément selon sa nature. Le modèle d'un élément de la partie opérative est ainsi composé de sous-modèles. Le sous-modèle modélisant le comportement générique de cet élément (*Element_comp*) correspond au Niveau Composant, tandis que le sous-modèle représentant les effets sur le fluide (*Element_cont*) correspond au Niveau Contextualisation. Toutefois, certains éléments comme ceux concernant l'instrumentation, n'agissent pas sur le fluide, contrairement aux éléments passifs (c'est-à-

dire non commandables) de tuyauterie. Ainsi, selon la nature de l'élément à représenter, son modèle peut être constitué uniquement d'une représentation Niveau Composant ou Niveau Contextualisation.

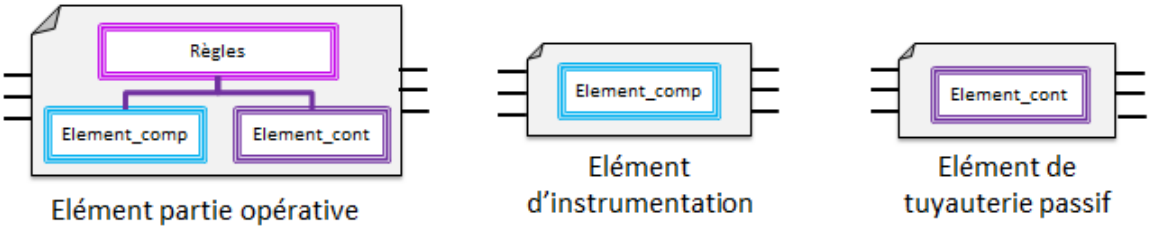


FIGURE 5.3 – Structuration du modèle d'un élément selon sa nature

Dans la section suivante, la proposition de structuration des modèles de simulation élémentaires sera appliquée pour construire un modèle de simulation de procédé.

5.3.2 Démarche de construction d'un modèle de simulation

Comme nous l'avons évoqué dans le chapitre 3, la construction du modèle de simulation suit un processus. La première phase permet d'obtenir le modèle conceptuel, la second phase concerne l'implémentation du modèle conceptuel pour obtenir le modèle de simulation.

Ainsi, en vue d'obtenir le modèle de simulation du procédé, il faut tout d'abord élaborer le modèle conceptuel. Dans ce modèle conceptuel apparaissent notamment les différents types d'éléments et les différentes connections entre les éléments. A partir d'un schéma de l'architecture fonctionnelle du procédé, les différents types d'éléments et les différentes connections entre ces éléments sont identifiés. Une fois les différents types d'éléments identifiés, on construit leurs modèles conceptuels. Puis, en instanciant ces modèles conceptuels génériques par rapport au schéma du procédé, est obtenu le modèle conceptuel du procédé. Le modèle de simulation du procédé se construit à partir de ce modèle conceptuel, en utilisant les modèles de simulation correspondant aux éléments qui ont été définis au préalable.

Dans le but de faciliter la construction des ces modèles de simulation, nous proposons la démarche suivante débutant par la construction du modèle conceptuel. On définit la structure du modèle de simulation selon sa nature (Etape 1), lors de la conception du modèle conceptuel. Le passage du modèle conceptuel vers le modèle de simulation se fait au travers d'une étape de spécification du comportement des modèles sous-jacents (Etape 2) puis de leur implémentation (Etape 3). Enfin, à partir du modèle conceptuel de l'élément et des modèles de simulation sous-jacents, le modèle de simulation de l'élément est implémenté (Etape 4).

5.3.3 Application de la proposition

Afin d'illustrer notre proposition, nous avons retenu le cas d'étude représenté sur la Figure 5.4. Le système est constitué de deux soutes (St1 et St2) séparées par une vanne motorisée à deux voies (VM1). La soute St2 est reliée à une vanne motorisée (VM2). Chaque vanne est constituée d'un capteur de fin de course d'ouverture (FcO) et de fermeture (FcF).

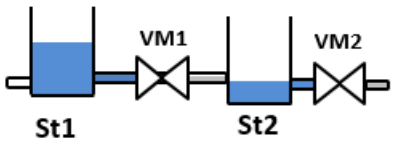


FIGURE 5.4 – Cas d'étude

Pour obtenir le modèle de simulation de ce système, nous utiliserons une approche orientée objet. Les modèles de simulation seront par ailleurs exprimés dans le langage Modelica, qui en plus d'être orienté objet permet une représentaton multi-domaine. En premier lieu, nous nous intéresserons à l'obtention des modèles de la vanne et de la soute. Nous détaillerons en particulier la construction du modèle de simulation de la vanne (cas le plus complet). Puis, nous nous intéresserons à la construction du modèle de simulation du système à partir des modèles précédemment obtenus.

5.3.3.1 Modélisation de la vanne

La vanne peut être vue comme un élément de la partie opérative. De ce point de vue, le modèle conceptuel de son comportement peut être représenté sous la forme d'un automate à état fini (Figure 5.5). Dans cette modélisation, la prise en compte d'une commande d'ouverture ou de fermeture se fait uniquement lorsque la vanne est en position fermée ou ouverte. Ainsi, lorsque la vanne est fermée (état 0 sur la Figure 5.5) et qu'elle reçoit une commande d'ouverture ($CmdO$), la vanne entame sa phase d'ouverture (état 1). La vanne ayant entamé sa manoeuvre elle quitte la position fermée (état 2). A la fin du temps de manoeuvre (*finOuverture*), la vanne est en position ouverte (état 3), laissant passer le fluide. La vanne étant ouverte, lorsqu'elle reçoit une commande de fermeture ($CmdF$) passe dans l'état 4. Elle entame sa phase de fermeture passant ainsi dans l'état 5. Puis, à la fin du temps de manoeuvre de fermeture, la vanne est dans la position fermée (état 0), empêchant le fluide de passer.

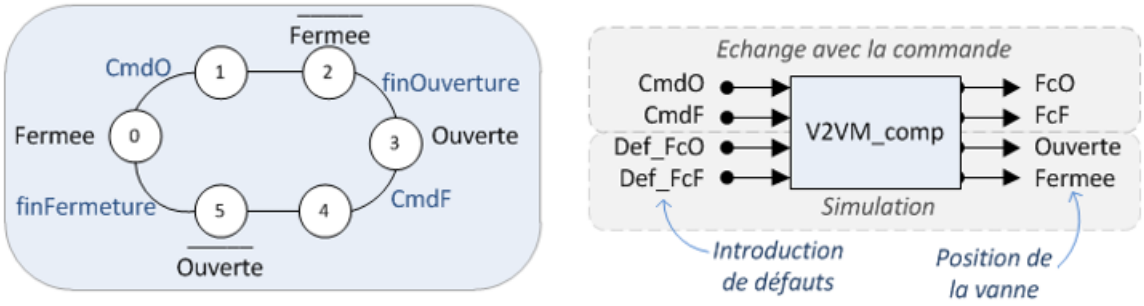


FIGURE 5.5 – Modèle conceptuel d'une vanne motorisée à deux voies

A noter que nous considérons qu'il peut y avoir un défaut sur les capteurs de fin de courses, l'introduction de ces défauts se fait au travers des variables Def_FcO (capteur de fin de course ouvert) et Def_FcF (capteur de fin de course fermée). Lors d'un fonctionnement normal (sans défaut), le capteur de fin de course de fermeture (FcF) prend la valeur `VRAIE` quand la vanne est en position fermée (état 0). De même, le capteur de fin de course d'ouverture (FcO) prend la valeur `VRAIE` lorsque la vanne est en position ouverte (état 3).

En appliquant le principe de structuration pour un élément de la partie opérative, on

obtient le modèle représenté à droite sur la Figure 5.6. L'équivalence de cette structuration dans le langage Modelica est donnée sur la partie gauche de la Figure 5.6.

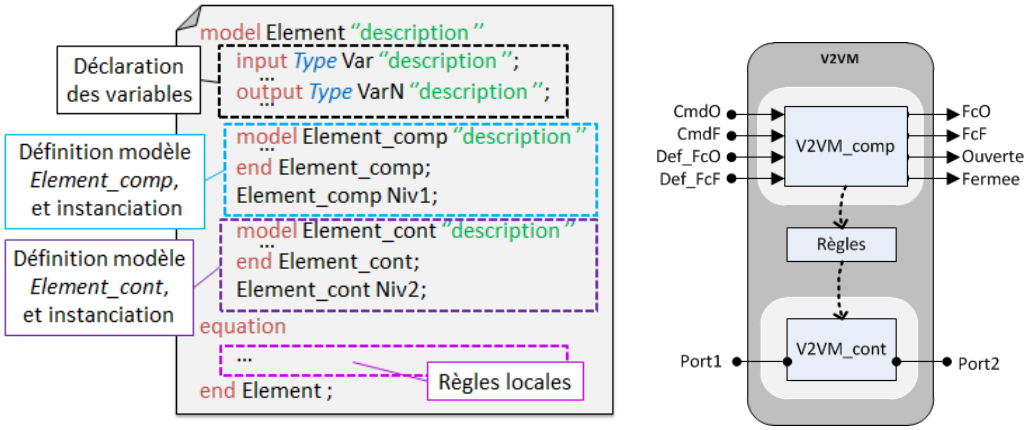


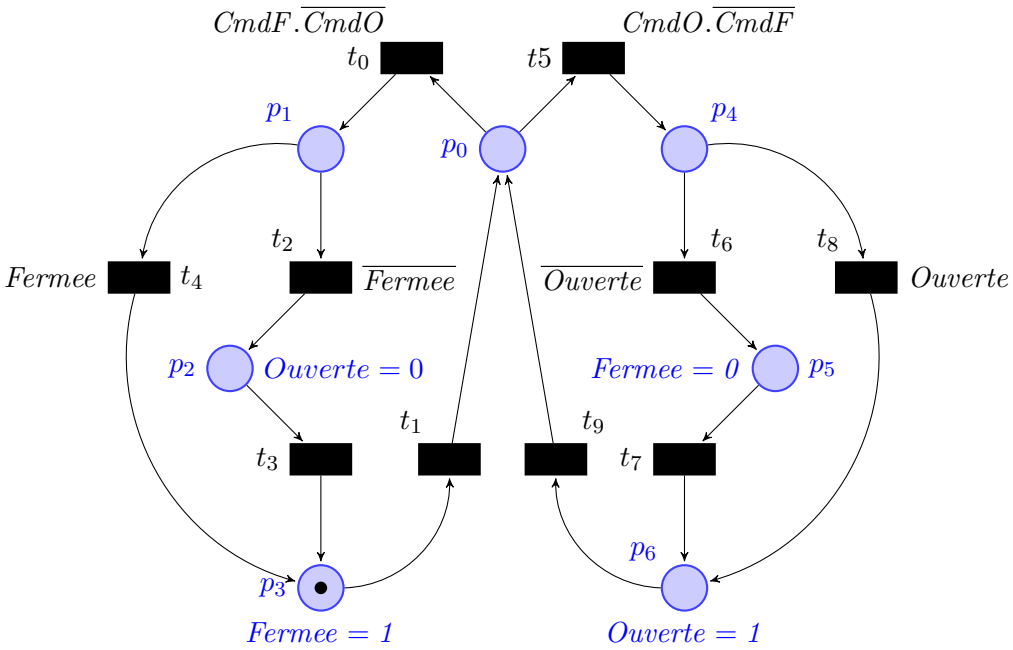
FIGURE 5.6 – Structure du modèle de simulation d'une vanne

Maintenant que la structure hiérarchisée du modèle de la vanne est définie, nous pouvons nous intéresser aux modèles sous-jacents. Commençons par la modélisation au Niveau Composant.

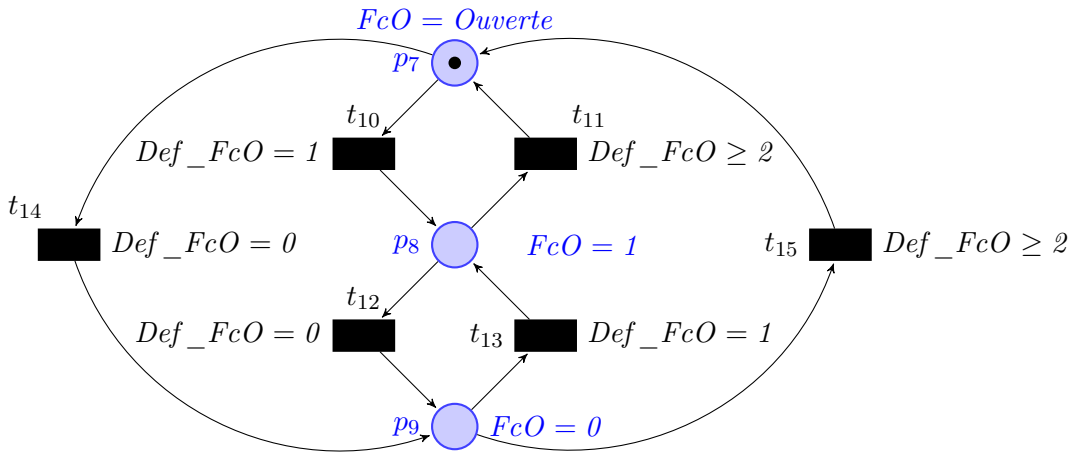
Si l'on reprend l'exemple donné dans [Prat et al., 2015], le comportement de la vanne peut être par exemple spécifié par un Réseau de Petri (Figure 5.7a). D'après le marquage du RdP, la vanne est dans la position fermée, la variable booléenne associée à cette position contient ainsi la valeur `VRAIE`. La transition `t1` est sensibilisée dès lors que la place `p3` est marquée. La condition associée au franchissement de cette transition est donc toujours vraie (il en va de même pour la transition `t9`). Une fois que cette transition est franchie, la place `p0` devient marquée. La vanne est alors en attente d'une commande d'ouverture ou de fermeture. Lors d'une commande de fermeture, la transition `t0` est franchie. La place `p1` reçoit le jeton de la place `p0`. La vanne étant déjà en position fermée, les transitions `t4` puis `t1` sont franchies. La place `p0` est à nouveau marquée. Lors d'une commande d'ouverture, la transition `t5` est franchie. La vanne n'étant pas déjà ouverte, la transition `t6` est franchie, la place `p5` devient alors marquée. A la fin du temps de manoeuvre, elle se retrouve ainsi en position ouverte, la place `p6` devenant alors marquée.

Outre l'évolution de la position de la vanne, nous devons représenter celle de l'état des capteurs de fin de course en fonction des défauts introduits (capteur *collé à 0* ou *collé à 1*). Une modélisation de l'évolution des valeurs des capteurs de fin de course, tenant compte de l'introduction de défauts, peut se faire comme illustré sur la Figure 5.7b, pour le capteur de fin de course d'ouverture. La place `p7` correspond à un état où le capteur de fin de course d'ouverture (`FcO`) fonctionne normalement. Ainsi lorsque cette place est marquée, la variable `FcO` prend la valeur de la variable `Ouverte`. Lorsque la place `p8` est marquée le capteur est « collé à 1 », et lorsqu'il s'agit de la place `p9` il est « collé à 0 ».

Une fois le comportement spécifié (et validé), le modèle élémentaire au niveau composant peut être implémenté dans le langage cible. La figure 5.8 est un exemple d'implémentation, par rapport aux modèles de connaissances (à droite de la figure).



(a) Evolution des positions de la vanne



(b) Capteur de fin de course d'ouverture

FIGURE 5.7 – Comportement d'une vanne motorisée à deux voies

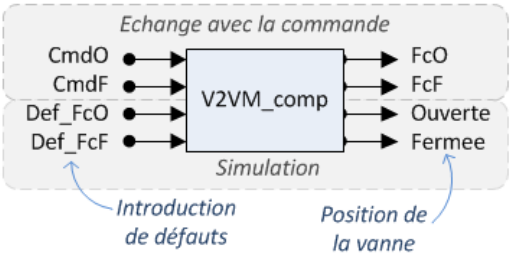
Passons, maintenant, à la modélisation de Niveau Contextualisation. Pour modéliser le comportement contextualisé de la vanne (les effets sur le fluide), on peut spécifier le comportement sous forme d'activités comme cela a été fait dans [Prat et al., 2015]. Si une représentation plus fine du comportement est nécessaire, on peut utiliser les lois de la physique. Par exemple : $debit = k * dp$, où k est la conductance hydraulique et dp la différence de pression aux bornes de la vanne.

Les librairies Modelica existantes proposent de tels modèles. Néanmoins, le niveau de détail utilisé pour représenter les phénomènes physiques est trop élevé pour nos besoins (de début de conception). En effet, très peu d'informations sur le système physique sont disponibles, en

particulier les informations liées au dimensionnement de la tuyauterie. Bien que ces modèles ne soient pas adaptés à notre besoin, la philosophie de modélisation utilisée peut tout de même être conservée pour construire les modèles de simulation de Niveau Contextualisaton.

```
model V2VM_comp
  /* Echanges avec la partie commande*/
  input Boolean CmdO;
  input Boolean CmdF;
  output Boolean FcO;
  output Boolean FcF;
  /* Variables rajoutee pour la simulation/*
  input Integer Def_FcO;
  input Integer Def_FcF;
  output Boolean Ouverte;
  output Boolean Fermee;
  /* Variables internes au modele/*
      ...
equation
      ...
  FcO = if Def_FcO >=2 then Ouverte
        elseif Def_FcO == 0 then false
        else true;
        ...
end V2VM_comp;
```

Modélisation évolution des positions de la vanne



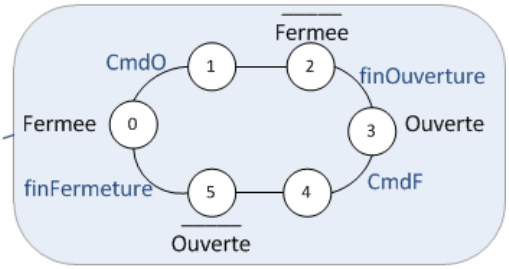
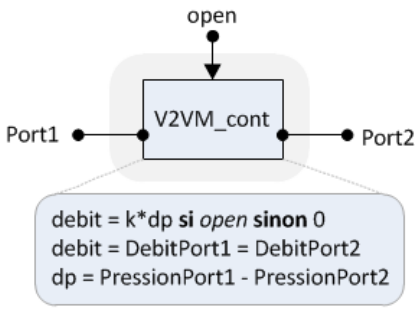


FIGURE 5.8 – Exemple de sous-modèle Composant pour une vanne

```
model V2VM_cont
  /* Fluide*/
  replaceable package Medium = Fluid.Medium;
  /* Variables échangées pour la simulation/*
  input Boolean open;
  FlowPort port1;
  FlowPort port2;
  /* Parametre d'initialisation/*
        ...
  /* Variables internes au modele/*
        ...
equation
        ...
  debit = if open then k*dp else 0;
  port1.vflow = debit /rho;
   /* Conservation de la masse/*
  port1.vflow + port2.vflow=0;
end V2VM_cont;
```




```
connector FlowPort
  /* Fluide*/
  replaceable package Medium=Fluid.Medium;

  Modelica.SIunits.Pressure  p;
  flow Modelica.SIunits.VolumeFlowRate vflow;

end FlowPort;
```

FIGURE 5.9 – Exemple de sous-modèle Contextualisation pour une vanne

Un exemple d'implémentation est donné Figure 5.9. Le modèle en langage Modelica est

représenté à gauche de l'illustration tandis que le modèle conceptuel est représenté en haut à droite. Afin de modéliser l'échange de flux (informationnel ou physique) entre des modèles, le langage Modelica propose une structure particulière : le *connecteur* (*FlowPort* sur la Figure 5.9). Un *connecteur* se définit à l'aide du mot-clé `connector`, comme l'illustre la Figure 5.10. Les variables à "connecter" y sont définies suivant le principe qui est représenté sur la partie droite de la Figure 5.10. Une fois, le connecteur défini, il peut ensuite être instancié dans les modèles des éléments, permettant par la suite de connecter les échanges de flux entre les différents modèles. La connexion des variables des différents modèles se fait ensuite à l'aide de l'équation **connect**(*variable* **,** *variable*).

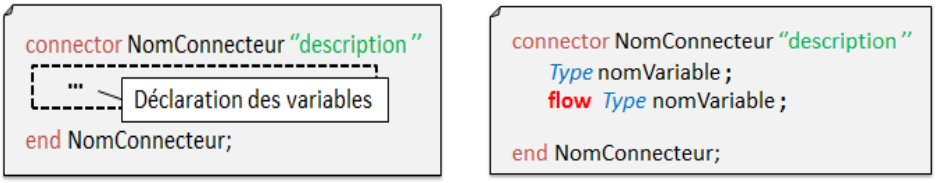


FIGURE 5.10 – Structure du modèle d'un connecteur

Ainsi, à partir du modèle conceptuel (à droite sur la Figure 5.11) et de l'implémentation des sous-modèles Niveau Composant et Contextualisation, on peut construire le modèle de simulation de la vanne. Une implémentation du modèle de simulation de la vanne est donnée à gauche sur la Figure 5.11.

```
model V2VM
  /* ** Declaration des variables ** */
        ...
  /* ** Sous-modeles  ** */
  model V2VM_comp
        ...
  end V2VM_comp;
  model V2VM_cont
        ...
  end V2VM_cont;
  /* Instanciation des sous-modèles */
  V2VM_comp  v2vm_comp ;
  V2VM_cont  v2vm_cont ;
equation
  /* **Connexion E/S sous-modeles** */
  connect(CmdO, v2vm_comp.CmdO);
        ...
  connect(port2, v2vm_cont.port2);
  /* **Regles** */
  v2vm_cont.open = not v2vm_comp.Fermee;
end V2VM;
```

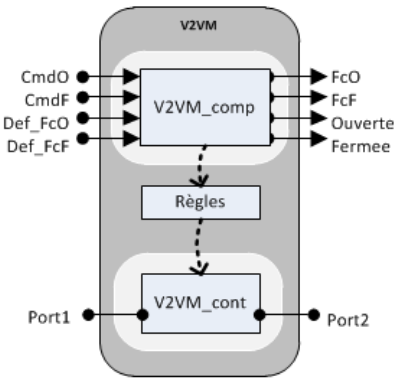


FIGURE 5.11 – Exemple de modèle de simulation de la vanne

Maintenant que nous avons obtenu le modèle de simulation de la vanne, nous nous intéresserons à la construction du modèle de simulation du système dans la section suivante.

5.3.3.2 Modélisation du système

Bien que disposant du modèle de simulation de la vanne, il nous faut disposer du modèle de simulation de la soute avant de pouvoir construire le modèle de simulation du système. La construction du modèle de simulation de la soute suit la même démarche que pour la vanne, à la différence qu'il est uniquement constitué du Niveau Contextualisation (cf partie gauche de la Figure 5.12). Le modèle de la soute dispose de deux connexions pour le remplissage et/ou la vidange (`Port1` et `Port2` sur la Figure 5.12). En plus, de ces deux flux physiques, il dispose également d'un flux informationnel (`Mes`) permettant de connecter un capteur de niveau le cas échéant. Pour construire le modèle conceptuel du système nous allons nous appuyer sur ceux de la vanne (Figure 5.11) et de la soute (Figure 5.12). Ainsi, le modèle conceptuel du système (Figure 5.13) est construit en instanciant les modèles conceptuels génériques d'une soute et d'une vanne.

```
model St
  /* ** Declaration des variables ** */
        ...
  /* ** Sous-modeles  ** */
  model St_cont
        ...
  end St_cont;
  St_cont  st_cont ;
equation
  /* **Connexion E/S sous-modele** */
        ...
  connect(port2, st_cont.port2);
end St;
```

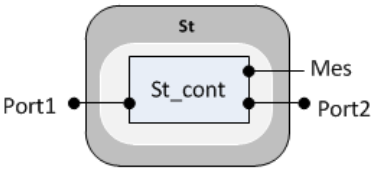


FIGURE 5.12 – Modèle de simulation et conceptuel d'une soute

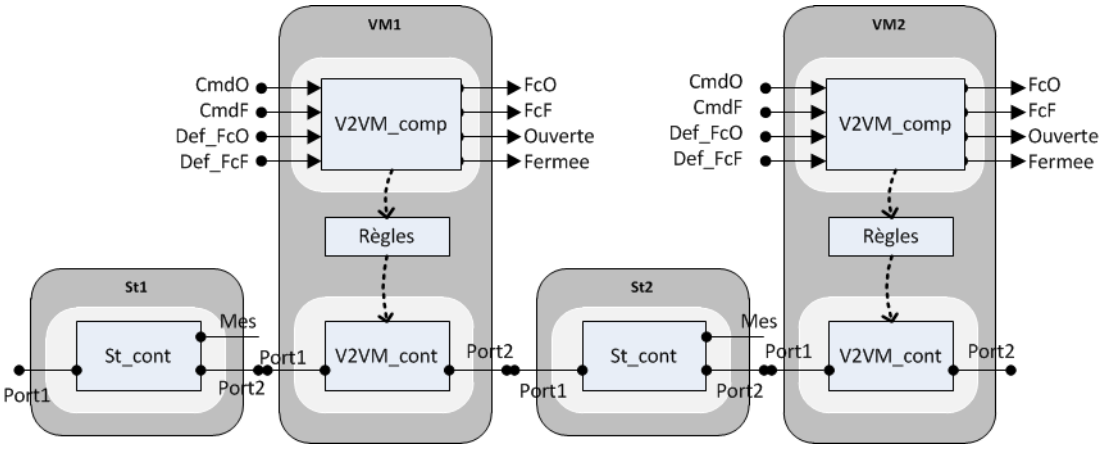


FIGURE 5.13 – Modèle conceptuel du système

Pour représenter la propagation du fluide dans le système, les variables représentant les points de connexion des flux physiques des éléments sont connectés. Ainsi, le `Port2` de la soute St1 est connecté au `Port1` de la vanne VM1, comme le montre la Figure 5.13. De même, le `Port2` de la vanne VM1 est connecté au `Port1` de la soute St2, et le `Port2` de la soute St2 est

connecté au `Port1` de la vanne VM2.

L'implémentation du modèle conceptuel se fait suivant le même principe. En effet, pour obtenir le modèle de simulation Figure 5.14, on instancie les modèles de simulation des éléments. La connexion entre les éléments pour les échanges de flux physique, se fait à l'aide de connecteurs.

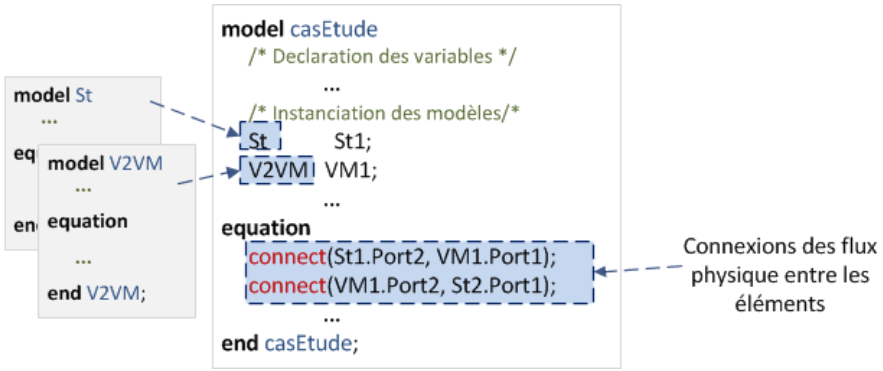


FIGURE 5.14 – Modèle de simulation du système

5.4 Bilan/Synthèse

Pour faciliter l'obtention de modèles de simulation tout au long de la conception, une stratégie de réutilisation doit être mise en place. La construction d'un modèle de simulation nécessitant des informations sur l'architecture fonctionnelle du procédé, un intérêt est porté sur la réutilisation des modèles métiers utilisés (schéma P&ID, par exemple). De plus, pour limiter l'effort de modélisation, nous nous pencherons également sur la réutilisation de modèles de simulation précédemment développés.

Dans un contexte de conception *orientée objet*, l'utilisation d'une approche orientée objet pour la modélisation et l'implémentation du modèle de simulation du procédé offre l'avantage de permette une réutilisation. Ainsi, le modèle de simulation du procédé peut être construit à partir des modèles de simulation des éléments du procédé, que l'on peut stocker dans une librairie de simulation. Néanmoins, les besoins de modélisation des éléments du procédé évoluant au fur et à mesure que le projet avance, il n'est pas possible de réutiliser les modèles des éléments "entiers" sur toute la durée du projet. En particulier à cause de la nature hétérogène du procédé (multi-domaine) et de l'évolution des besoins en terme de modélisation. Un modèle de simulation étant construit dans une intention particulière, pour le réutiliser l'objectif de simulation doit correspondre. En conséquence, plusieurs modèles de simulation d'un même élément doivent être construits pendant le projet, pour pouvoir effectuer les tests du système de conduite. Afin de réduire les temps de construction/re-construction de ces modèles, les modèles de simulation des éléments doivent être structurés de façon à pouvoir s'adapter aux besoins de modélisation (verrou 2).

Dans le cadre de la modélisation et de la simulation de système de gestion de fluide, nous avons proposé de construire les modèles de simulation des éléments, en décomposant la

modélisation du comportement de la partie opérative qui est plutôt à événements discrets, de celle du fluide dont la dominante est continue. Ainsi, les modèles des éléments sont construits via une approche modulaire :

- le Niveau Composant, où l'on représente le comportement générique d'un élément de la partie opérative ou d'instrumentation,
- le Niveau Contextualisation, où l'on représente les effets sur le fluide,
- le Niveau Règles, le plus abstrait, où l'on lie les sous-modèles Niveau Composant et Contextualisation.

A noter que selon la nature de l'élément du procédé à représenter, son modèle de simulation peut être composé uniquement d'un sous-modèle Niveau Composant (cas de l'instrumention qui n'agit pas sur le fluide) ou au contraire uniquement d'un sous-modèle Niveau Conxtextualisation (cas des éléments passifs de tuyauterie). Selon les besoins de modélisation, cette proposition de structure permet de réutiliser les précédents modèles de l'élément en ne modifiant qu'une partie du modèle (le sous-modèle concerné).

Nous avons appliqué la proposition de structuration des modèles de simulation sur un cas, afin d'en obtenir le modèle de simulation. Afin de réutiliser l'approche de génération du modèle de simulation du procédé sur toute la durée du projet, nous avons utilisé le langage de simulation Modelica. Outre sa nature hiérarchique et orienté objet, ce langage permet en effet une modélisation multi-domaine. Cela nous laisse ainsi envisager la possibilité de n'utiliser qu'un seul langage de simulation sur la durée du projet.

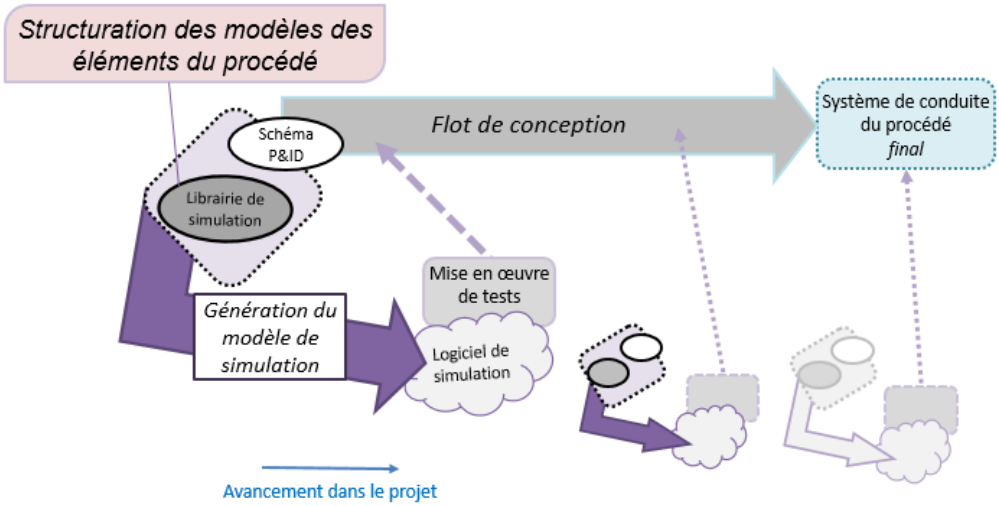


FIGURE 5.15 – Apport de la proposition pour permettre une approche de génération tout au long de la conception

Au travers de ce chapitre, nous avons posé les premiers principes en vue de faciliter l'obtention de la simulation, pour permettre la mise en place de tests en parallèle des activités de conception (Figure 5.15). Il s'agit maintenant de proposer une approche de génération automatisée de modèles de simulation, à partir d'une librairie de modèle de simulation des éléments et d'un schéma P&ID. Pour faciliter la réutilisation des modèles de simulation selon l'avancement dans le projet, nous avons proposé une manière de structurer les modèles

élémentaires, afin de faciliter la modélisation, levant ainsi le verrou 2. D'autre part, dans la perspective de réutiliser la même démarche de génération du modèle de simulation du procédé, nous envisageons l'utilisation du langage de simulation Modelica, et d'une librairie de simulation basée sur le concept de *vue*.

En vue de faciliter l'obtention de la simulation, dans le prochain chapitre nous nous intéresserons à l'automatisation de la génération du modèle de simulation du procédé.

6

Génération des modèles de simulation

Sommaire

6.1 Introduction

Dans le cadre des systèmes de production, il a été proposé de rajouter une phase de tests virtuels (*virtual commissioning*) avant la phase de tests réels. [Oppelt and Urbas, 2014] sont allés plus loin en proposant d'intégrer le *virtual commissioning* au processus de conception du système de commande (*integrated virtual commissioning*). L'avantage est de mener des tests de vérification des programmes de commande, en parallèle de l'avancement du travail de l'automaticien, afin de détecter de corriger les erreurs au plus tôt. Cependant, malgré une littérature riche en propositions, le *virtual commissioning* n'est pas encore devenu une pratique courante dans l'industrie. Ce serait dû au fait que ce type de vérification nécessite un effort important de modélisation ainsi qu'un important savoir-faire en simulation de la part de l'automaticien [Oppelt et al., 2014]. D'autre part, comme le souligne [Barth and Fay, 2013] l'obtention manuelle des modèles de simulation est très coûteuse en temps, d'autant plus qu'il faut maintenir à jour le modèle de simulation en fonction des modifications du système physique à commander et de la disponibilité des informations. La mise en oeuvre d'une approche de génération automatisée des modèles de simulation, à partir de modèles métiers du système physique considéré, semble être un moyen d'y parvenir.

Dans le cadre de la conception de système d'aide à la conduite de navire, nos travaux s'intéressent plus particulièrement à la conception de la chaîne de contrôle-commande des sous-systèmes du navire. Ces systèmes peuvent être considérés comme des systèmes de conduite de procédés. Il s'agit principalement de systèmes de gestion de fluides (par exemple, un système de production et distribution d'eau douce). Comme évoqué lors du chapitre précédent, une des difficultés est de pouvoir ré-utiliser les modèles en fonction de l'avancement et des vérifications que l'automaticien souhaite effectuer. Selon l'avancement du projet (et les informations disponibles) le niveau de détails dans la modélisation et les caractéristiques de l'exécution de la simulation peuvent être différents. De plus, lors de la modélisation d'un système de gestion de fluide, il faut prendre en compte le caractère continu du comportement du fluide, en plus de celui de la partie opérative qui peut être considéré comme un comportement à événements discrets. Nous avons donc proposé une stratégie de réutilisation basée sur une librairie de modèles de simulations des éléments du procédé, ainsi qu'une structuration de ces modèles distinguant la modélisation de la partie opérative des effets sur le fluide (cf Chapitre 5).

Afin de réduire les coûts et les délais liés au développement des modèles de simulation, nous visons la réutilisation des données d'ingénierie qui concerne tant les informations liées au procédé à modéliser, que celles liées au système de commande par rapport aux signaux d'entrées/sorties qui doivent être échangés. De plus, dans un contexte Ingénierie Dirigée par les Modèles, il semble judicieux que la génération des modèles de simulation s'intègre dans une démarche de conception automatisée de contrôle-commande.

Ainsi, pour intégrer des vérifications par simulation tout au long de la conception d'un système de conduite de procédé, nous visons la définition d'une approche de génération automatisée de modèles de simulation s'intégrant dans un flot de conception automatisé de contrôle-commande (verrou 3 identifié au chapitre 3). Suite à l'état de l'art réalisé dans la section 6.2, nous proposons dans la section 6.3 un flot de génération automatisée de modèle de simulation.

6.2 Etat de l'art

En vue de construire notre proposition pour une obtention automatisée des modèles de simulation, nous nous intéresserons dans un premier temps aux approches de modélisation pour la simulation. Puis, nous nous intéresserons aux travaux concernant la génération automatisée de modèles de simulation pour tester les systèmes de commande de procédé. Enfin, pour tirer pleinement avantage de la génération automatisée, nous nous intéresserons à la génération automatisée de modèles de simulation dans un flot de conception automatisé de contrôle-commande.

6.2.1 Approches de modélisation pour la simulation

Une classification des différentes méthodes de modélisation de la simulation est proposée par [Li et al., 2013], qui distinguent trois catégories. La première correspond aux mé-

thodes de modélisation unifiées. Elle contient notamment les méthodes qui utilisent un langage permettant la représentation multi-domaine. Avec par exemple le langage Modelica qui permet de décrire différents types de systèmes physiques via des équations mathématiques, [Elmqvist et al., 2001]. Ce langage concerne la simulation de systèmes à temps continu ou hybride. On y retrouve aussi les méthodes qui utilisent un langage basé sur plusieurs formalismes différents. Avec par exemple le langage UML (Unified Modelling Language), ou encore le formalisme DEVS (Discrete Event System Specification), [Zeigler et al., 1995].

Une deuxième catégorie correspond aux méthodes de modélisation combinée qui permettent de composer des modèles de formalismes différents et de les simuler intégralement. On y retrouve la co-simulation qui consiste à utiliser différents systèmes de simulations (simulateurs) qui s'échangent des données de simulation Et la multi-modélisation qui concerne la composition de modèles de formalismes différents au niveau sémantique. Simulink ou Ptolemy II sont des exemples d'outils permettant la multi-modélisation.

La dernière catégorie correspond aux méthodes de modélisation basée sur l'IDM qui permettent une modélisation formelle de modèles indépendamment de leur implémentation, puis de générer à partir de ces modèles le code exécutable. L'IDM est issue du génie logiciel, pour plus d'informations à ce sujet le lecteur peut se réfférer aux travaux de [Bézivin, 2005], ou [Hutchinson et al., 2011].

Il est également possible d'utiliser des combinaisons des méthodes des différentes catégories identifiées. Afin d'obtenir les modèles de simulation par génération automatique, les méthodes de modélisation basée sur l'IDM semblent bien adaptées. D'un autre côté, pour représenter le comportement hybride du procédé, les méthodes utilisant un langage de modélisation multi-domaine semblent appropriées. Ainsi pour répondre à nos besoins il semble judicieux d'utiliser une approche basée sur l'IDM avec une méthode utilisant un langage de modélisation multi-domaine tel que le langage Modelica. L'utilisation du langage Modelica a d'ailleurs été retenue dans divers travaux tant industriels que académiques.

6.2.2 Génération automatique de modèles de simulation

Afin de faciliter l'obtention et le maintien à jour des modèles de simulation, il semble intéressant d'utiliser comme source de données d'entrées un modèle métier représentant l'architecture fonctionnelle du procédé et qui est utilisé tout au long du projet. Tel est le cas du schéma de la tuyauterie et de l'instrumentation (P&ID), qui sert aussi pour concevoir le système de contrôle-commande de procédé, dont l'interface de supervision, [Hoernicke et al., 2015].

Différents travaux se sont ainsi basés sur le P&ID pour obtenir automatiquement les modèles de simulation du procédé afin de pouvoir ensuite vérifier le système de contrôle-commande. On peux citer par exemple, les travaux de [Barth and Fay, 2013] ou ceux de [Arroyo et al., 2016]. La génération se fait à partir du P&ID du procédé, et permet d'obtenir des modèles dans le langage Modelica. Toutefois, la mise en oeuvre de la génération automatique des modèles de simulation n'est pas détaillée, seul les principes y sont présentés.

[Oppelt et al., 2014] ont proposé un mécanisme pour générer les modèles de simulation à

partir des données d'ingénierie, en cinq étapes. La première étape qui est réalisée par l'expert en simulation est la *création des objets de simulation*, cela correspond à la création d'une librairie de modèles élémentaires. La deuxième étape vise à étendre le modèle métier source pour y intégrer des données supplémentaires utilisées pour la simulation. Dans la troisième étape on effectue un mapping entre les modèles élémentaires et leurs équivalents dans le modèle métier. La quatrième étape quant à elle correspond à la collecte et l'échange des données pour la création du modèle de simulation. Enfin, la dernière étape permet de générer le modèle de simulation.

La généricité du mécanisme présenté par [Oppelt et al., 2014] est intéressante car il semble pouvoir être implémenté indépendamment du logiciel de simulation cible et/ou de l'approche de modélisation utilisée pour la simulation. Cependant, la formalisation du mécanisme reste encore à un niveau relativement abstrait.

Les travaux dans la littérature restent très discrets sur la formalisation du processus de génération automatique des modèles de simulation de procédé. Afin de permettre l'utilisation d'une telle approche dans un milieu industriel, il ressort donc un besoin de formaliser explicitement un flot de génération de modèles de simulation, à partir d'un P&ID.

6.2.3 Simulation et flot de conception automatisé

Dans le cadre des systèmes manufacturiers, plusieurs travaux se sont intéressés à la génération conjointe de programmes de commande et de modèles de simulation. [Lallican et al., 2007] génèrent à partir d'un modèle du système transitique, la commande et le modèle de simulation de la partie opérative. La simulation conjointe y est utilisée pour vérifier la commande générée. Le logiciel SimSED est utilisé pour exécuter le modèle de la partie opérative, le code de commande est quant à lui exécuté sur la machine virtuelle du logiciel STRATON. La génération se fait à partir d'une description du système, exprimée dans un langage spécifique au domaine (DSL) et d'une bibliothèque de composants basée sur le concept de *vue*. Les vues *parties opératives* des composants sont utilisées pour obtenir, par transformation de modèle, le modèle de simulation de la partie opérative du système transitique. Ces travaux ont étés étendus par [Bévan, 2013] dans le cadre de systèmes transitiques reconfigurables.

Afin d'aider au pilotage d'un système de production, par l'utilisation de la simulation en ligne, [Adam et al., 2012] génère conjointement la commande et un observateur basé sur un simulateur à événement discrets. Cet observateur a pour but d'initialiser le simulateur avec l'état réel du système.

Dans les travaux de [Lallican et al., 2007, Bévan, 2013, Adam et al., 2012], on constate que la génération conjointe du modèle de simulation et du programme de commande, se fait à l'aide d'une bibliothèque basée sur le concept de *vue*. Cependant les modèles de simulation générés ne permettent pas la prise en compte du caractère continu du comportement du fluide.

Dans le cadre des systèmes de contrôle de procédé, différentes démarches outillées, basées sur l'IDM, ont été proposées pour générer le système de commande. On retrouve par exemple, les travaux concernant le projet AUTOKON, avec notamment la définition d'un

profil UML spécifique *UML Automation Profile* (UML AP) [Vepsäläinen et al., 2008]. Dans [Vepsäläinen and Kuikka, 2014] est proposé une démarche basée sur UML AP où il est généré les modèles de simulation des séquences de commande. Cette approche de test est de type Model in the loop, elle ne permet donc pas de tester le programme de commande réel.

Au lieu de chercher à adapter les outils de développement logiciel aux autres domaines, [Bignon et al., 2013] proposent d'utiliser les outils des experts de ces autres domaines comme données d'entrée pour les activités de programmation. Ainsi, la démarche outillée Anaxagore permet de générer automatiquement programmes de commande et interfaces de supervision, à partir du modèle P&ID du système (cf Figure 6.1).

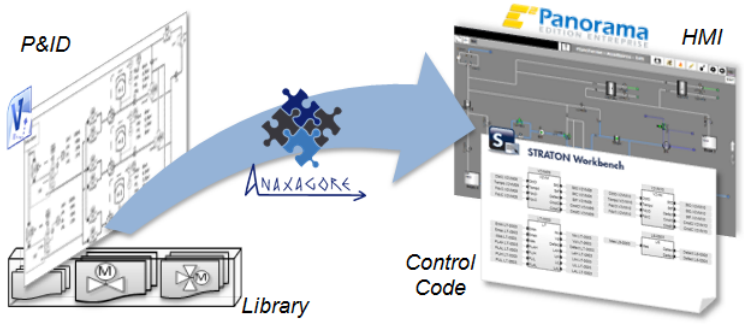


FIGURE 6.1 – Principe de la démarche Anaxagore

Afin de proposer une démarche générique, le P&ID qui est spécifique à une plateforme, est transformé en un modèle indépendant de toute plateforme : le synoptique. La génération se fait à partir de ce modèle, et d'une bibliothèque d'éléments standards basée sur le concept de *vue*. D'autre part, un mapping est effectué automatiquement à travers la *nomenclature*. Ce modèle comporte la liste de toutes les instances des éléments du système ainsi que les différentes variables qui y sont associées. Grâce à un attribut, on peut savoir à quelle vue est rattachée la variable. De plus, la *nomenclature* contient toutes les connexions entre les éléments qui figurent sur le synoptique. La nomenclature est une synthèse des informations contenues dans le P&ID, enrichies par des informations complémentaires liées aux variables échangées entre supervision, commande et procédé. C'est ce modèle qui est utilisé tout au long du flot de conception pour la génération. Toutefois cette démarche n'intègre pas la génération des modèles de simulation.

6.2.4 Bilan

Afin d'apporter un support aux concepteurs du système de contrôle-commande, il faut que la génération des modèles puisse être utilisée tout au long du projet (notion de virtual commisssioning integré). Les modèles de simulation des éléments doivent par conséquent pouvoir être enrichis en fonction des besoins (par exemple, en fonction de ce que l'on souhaite vérifier). Pour y parvenir, une solution consiste à utiliser un langage de simulation qui soit orienté-objet et permettant une modélisation multi-domaine (tel que le langage Modelica). L'avantage est que l'on peut réutiliser des modèles des librairies existantes, les modifier ou encore en créer

de nouveaux. L'utilisation du langage Modelica a d'ailleurs déjà été implementée avec succès, comme par exemple dans les travaux de [Barth and Fay, 2013]. Cependant, les travaux de génération automatique de modèles Modelica à partir de P&ID manquent de formalisation. L'utilisation d'une approche basée sur l'IDM devrait pouvoir permettre de combler ce manque.

Ce type d'approche a en outre été mise en oeuvre dans le cas de systèmes à événements discrets. Notamment dans les travaux [Lallican et al., 2007, Bévan, 2013, Adam et al., 2012], où la génération est intégrée dans le flot de conception automatisé du programme de commande du système manufacturier. Pour y parvenir, ils utilisent une bibliothèque basée sur le concept de *vue*. De plus, ce concept de *vue* est aussi utilisé dans [Bignon et al., 2013], où le flot de conception automatisé permet d'obtenir programmes de commande et IHM de supervision à partir du P&ID. En se basant sur ces travaux, il devrait être possible de proposer un flot de génération de modèles de simulation Modelica du procédé.

Par l'utilisation d'une approche basée sur l'IDM, et d'une bibliothèque basée sur le concept de *vue*, nous devrions donc être en mesure de générer les modèles de simulation Modelica du procédé (Figure 6.2). D'autant plus que l'utilisation du concept de *vue* permettrait d'avoir dans la bibliothèque différentes *vues* de simulation, en fonction de l'avancement du projet, et donc des besoins en termes de vérifications. Par la suite, la proposition pourrait être intégrée dans un flot de conception automatisé, permettant ainsi d'apporter un réel support aux concepteurs pour tester les programmes de commande.

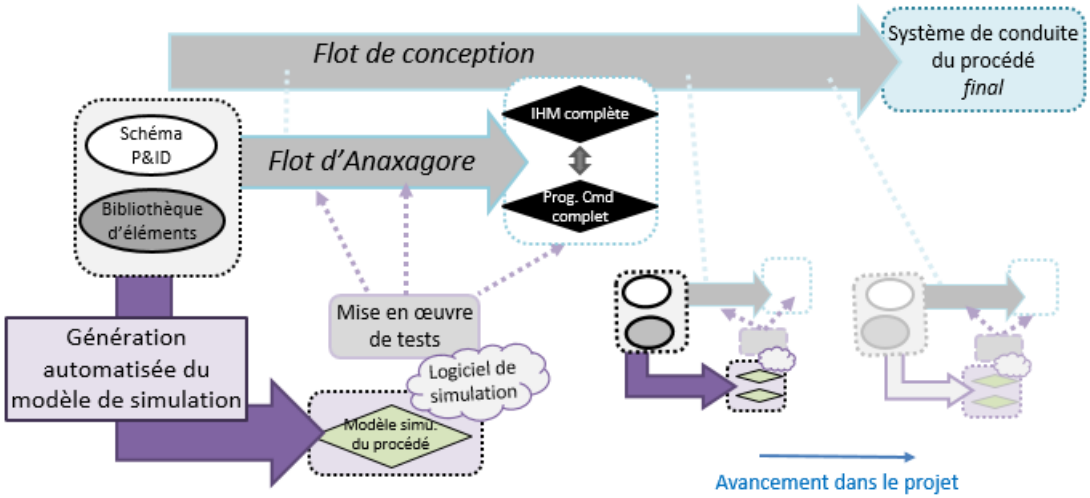


FIGURE 6.2 – Approche de génération automatisée envisagée

6.3 Proposition d'un flot de génération automatisé

6.3.1 Principe de la proposition

Suite à la revue de la littérature, nous proposons de générer les modèles de simulation dans le langage cible *Modelica*, à partir d'une bibliothèque d'éléments et d'un schéma P&ID, en parallèle des activités de l'automaticien. Notre proposition est inspirée de la démarche

outillée Anaxagore (issue des travaux de [Bignon et al., 2013]). En particulier, nous retenons le diagramme synoptique comme modèle d'entrée qui a l'avantage d'être une représentation générique du schéma P&ID. Ce modèle est indépendant de la plateforme sur laquelle a été saisi le schéma P&ID. D'autre part, la bibliothèque d'éléments standards, basée sur le concept de vue, a été étendue par une vue de simulation qui contient les modèles de simulation élémentaires.

Notre proposition consiste à générer le modèle de simulation en langage *Modelica* ainsi que la libraire de simulation associée (elle aussi en langage *Modelica*). Cette librairie de simulation contient tous les modèles élémentaires qui doivent être instanciés. Elle est notamment requise pour pouvoir exécuter le modèle de simulation du procédé dans un logiciel *Modelica*. L'avantage est ici d'importer dans le logiciel uniquement les modèles élémentaires de simulation qui sont utilisés.

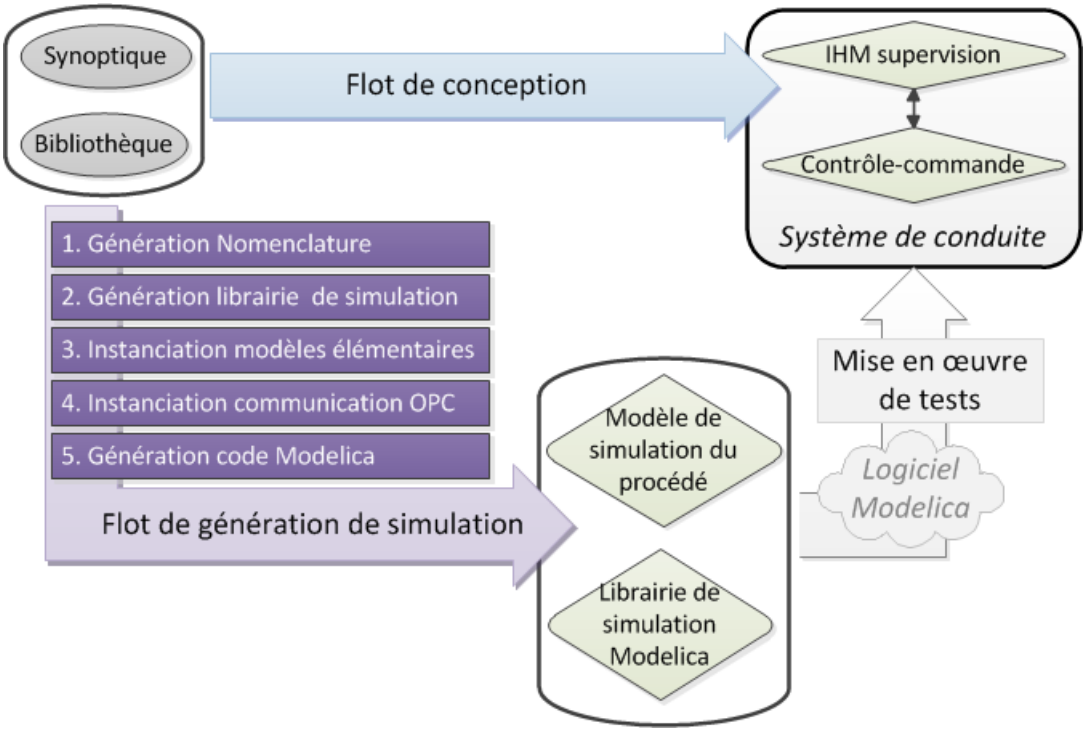


FIGURE 6.3 – Principe de la proposition

Afin de générer le modèle de simulation et la libraire *Modelica* associée, cinq étapes ont été définies (Figure 6.3). La première étape est la *Génération de la Nomenclature*, elle permet de réaliser le mapping entre les différentes vues et les informations contenues dans le Synoptique. Nous reprenons pour cela le concept de la *Nomenclature* de [Bignon et al., 2013]. La deuxième étape correspond à la *Génération de la Librairie de simulation* dans le langage *Modelica*. Puis, au travers des trois dernières étapes on va générer le modèle de simulation. Ainsi, la troisième étape correspond à l'*Instanciation des modèles élémentaires* et de leurs connexions, tandis que la quatrième étape concerne l'*Instanciation de la communication OPC* pour les échanges avec

le système de commande. Enfin la cinquième et dernière étape, *Génération du code Modelica*, permet de générer le modèle de simulation complet qui sera chargé sur le simulateur.

Ces étapes sont implémentées au travers d'un flot de génération, représenté sur la Figure 7.1. A partir de la Bibliothèque d'éléments et du Synoptique, la Nomenclature est générée (étape 1). Outre le mapping entre les vues de la Bibliothèque, la Nomenclature contient une synthèse des informations relatives aux instances des éléments. Ainsi, toutes les variables échangées entre le système de contrôle/commande et le procédé sont répertoriées ainsi que toutes les connexions (flux physiques et informationnels) entre les instances des éléments contenues dans le Synoptique. En conséquence, ce modèle est utilisé pour la génération de la Libraire *Modelica* et du modèle de simulation.

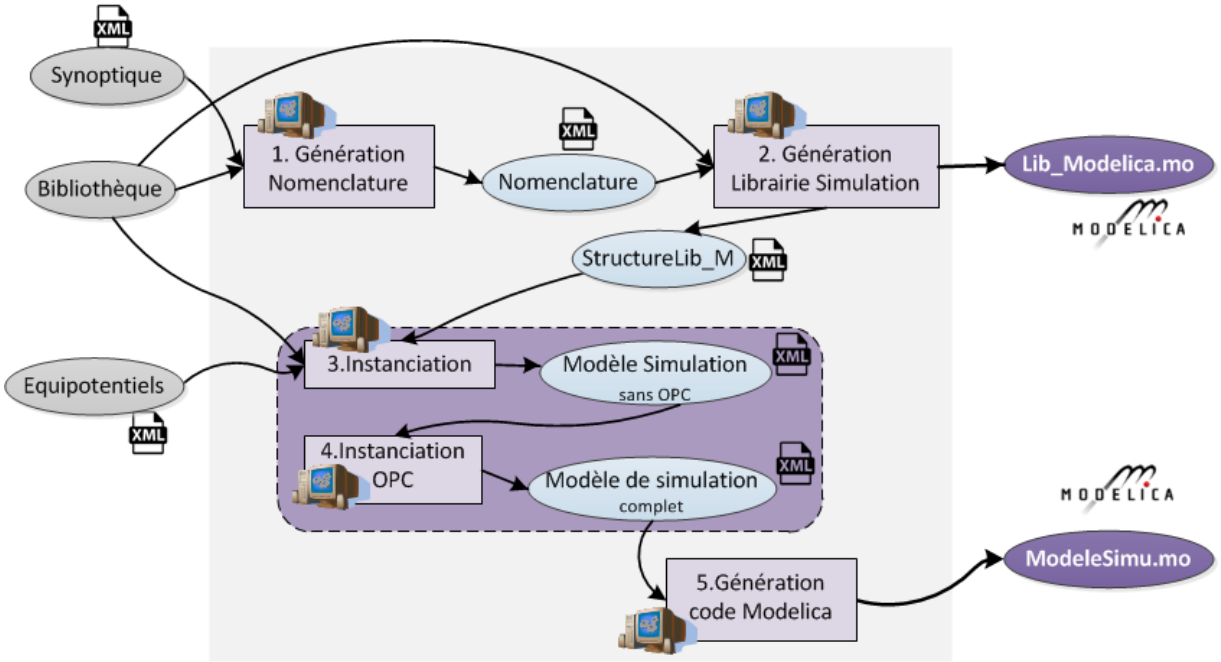


FIGURE 6.4 – Flot de génération de modèles de simulation

La Nomenclature est ensuite utilisée, avec la Bibliothèque d'éléments, pour la génération de la Librairie *Modelica* (étape 2). Cette librairie contient tous les modèles de simulation élémentaires qui doivent être instanciés pendant la génération du modèle de simulation du procédé. En plus de la Librairie Modelica, la structure de la librairie est aussi générée (*StructureLib_M*). Elle contient, en plus de l'arborescence de la librairie, tous les emplacements des modèles de simulation élémentaires nécessaires pour la phase d'instanciation.

Les deux étapes suivantes concernent la construction du modèle de simulation. Cette phase de construction est réalisée à l'aide de modèles de simulation intermédiaires au format XML. Dans un premier temps, les modèles des éléments sont instanciés avec leurs connexions (étape 3). L'instanciation des modèles est faite à partir de la Nomenclature et de StructureLib_M. Pour instancier les connexions entre les modèles de simulation (instanciés), la Nomenclature est utilisée avec des règles génériques issues des vues de simulation. Un autre

modèle d'entrée est aussi utilisé, le modèle des Equipotentiels qui représente les points de connexion à un même potentiel. Pour permettre la communication entre le procédé simulé et le système de contrôle/commande, l'instanciation de la communication OPC est faite au travers de l'étape 4. A partir de la Nomenclature, les variables OPC sont instanciées. Ensuite, celles-ci sont ajoutées au *Modèle de Simulation sans OPC* pour obtenir le *Modèle de Simulation Complet*. Finalement, la dernière étape transforme le *Modèle de Simulation Complet* en code *Modelica* (*ModeleSimu.mo*). Le modèle de simulation dans le langage *Modelica* est ainsi généré.

Par la suite, le flot de génération des modèles de simulation sera présenté en détail. Une première partie sera consacrée à la génération de la Librairie Modelica, et une seconde partie à la génération du modèle de simulation.

6.3.2 D'un modèle métier à la génération d'une libraire Modelica

Comme évoqué précédement, la génération de la Nomenclature (étape 1) bassée sur les travaux de [Bignon et al., 2013], se fait à partir du Synoptique et de la Bibliothèque d'éléments standards. Le méta-modèle du Synoptique n'ayant pas subi de modification, nous présentons uniquement les modifications apportées au méta-modèle de la Bibliothèque d'éléments standards.

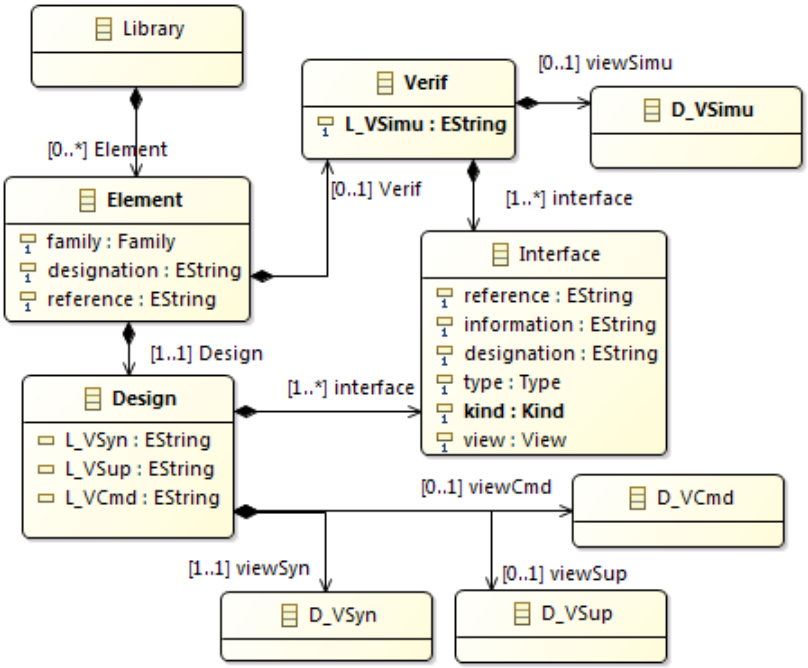


FIGURE 6.5 – Extension du métamodèle de la Bibliothèque de [Bignon et al., 2013]

Le méta-modèle de la Bibliothèque (cf Figure 6.5) a été modifié de la façon suivante. Un élément (*Element*) contient ainsi une partie liée à la conception et une liée à la vérification (*Design* et *Verif* sur la Figure 6.5). Les chemins vers les vues (*D_ VSyn, D_ VSup, D_ VCmd*)

de conception sont contenus dans les attributs (`L_VSyn, L_VSup,L_VCmd`). Le chemin vers la vue de simulation (`D_VSimu`) est contenu dans l'attribut `L_VSimu` de *Verif.* La liste des valeurs possibles de l'attribut `kind` pour une *Interface* a été étendue pour prendre en compte deux grandes catégories de variables échangées : celles qui sont internes à la simulation et celles qui sont externes.

Les variables internes à la simulation peuvent être séparées en trois catégories. Une première catégorie de variables internes à la simulation correspond à celles qui sont échangées à l'intérieur du modèle de simulation. C'est le cas par exemple des flux physiques et informationnels qui sont échangés entre les modèles des différents éléments composant le procédé. Une seconde catégorie de variables internes concerne des variables d'entrée qui sont rajoutées pour les besoins de l'utilisation de la simulation. Elles permettent par exemple d'introduire des défauts (aléas). La dernière catégorie de variables internes concerne les variables de sortie rajoutées pour permettre et/ou faciliter l'analyse des résultats de la simulation. Les variables externes à la simulation correspondent quant à elles aux variables échangées avec la partie de contrôle-commande. Ainsi, les variables d'entrée de la simulation concernent les commandes qui sont reçues, tandis que les variables de sortie concernent les informations qui sont envoyées vers le système de commande. Ce sont donc ces variables externes qui sont utilisées lors de la communication OPC.

Pendant la *Génération de la Nomenclature* (étape 1), toutes les variables de chaque élément instancié dans le Synoptique sont collectées. En conséquence, la Nomenclature contient toutes les variables de simulation, externes et internes, associées à chaque instance des éléments du procédé (désignées au niveau du Synoptique). La Nomenclature est ensuite utilisée pour la *Génération de la Librairie Modelica* (étape 2). En effet, c'est à partir de celle-ci, que la liste des modèles de simulation, qui doivent être contenus dans la Librairie Modelica, est définie.

Pour créer la Librairie Modelica, en premier lieu il est construit le modèle de la structure de librairie (*StructureLib_M*). Puis, à partir des vues de simulation de la Bibliothèque, les modèles correspondant sont collectés, et insérés dans le modèle *StructureLib_M* afin d'obtenir le modèle de la Librairie. Pour ces deux modèles, nous avons défini un métamodèle commun (Figure 6.6). La librairie de simulation (*Librairie* sur la Figure 6.6) est composée d'un ou plusieurs "Package" qui peuvent, le cas échéant, eux-même contenir un ou plusieurs "Package". Un "Package" est référencé par un attribut `name` pour son nom, et un attribut `description` pour renseigner ce qu'il contient. D'autre part, les modèles de simulation, correspondant aux "Component", sont encapsulés dans un ou plusieurs "Package". L'attribut `name` d'un "Component" fait référence au nom du modèle de simulation, et l'attribut `dirSimu` à la localisation de la vue de simulation.

Ensuite, le modèle de la Librairie, qui est un fichier xml, est converti en fichier Modelica. On obtient ainsi la Librairie Modelica (*Lib_Modelica.mo*), qui contient tous les codes sources des modèles de simulation des éléments qui doivent être instanciés. Néanmoins, le code source étant utilisé uniquement durant la compilation du modèle de simulation complet, c'est le modèle *StructureLib_M* qui sera utilisé pour l'instanciation (étape 3).

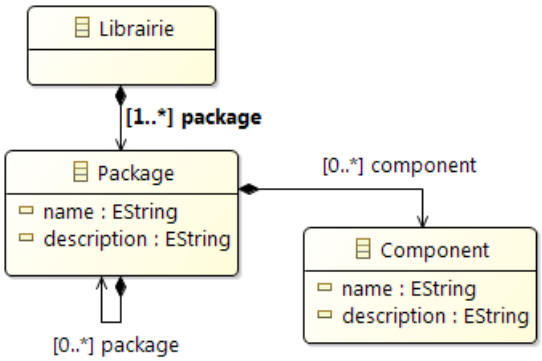


FIGURE 6.6 – Extrait du Métamodèle de la librairie de simulation

6.3.3 La génération du modèle de Simulation

La Librairie Modelica ayant été générée, la génération du modèle de simulation peut commencer. La première phase consiste à construire le modèle de simulation (étape 3 et 4), tandis que la seconde traduit le modèle de simulation en code *Modelica* (étape 5). Un métamodèle pour les modèles de simulation a été défini. Ce métamodèle permet de définir aussi bien les modèles de simulation intermédiaires que le modèle de simulation final en langage Modelica. Il est ainsi nécessaire, avant de s'intéresser plus en détail à la génération du modèle de simulation, de présenter ce métamodèle puisqu'il est commun pour tous les modèles de simulation.

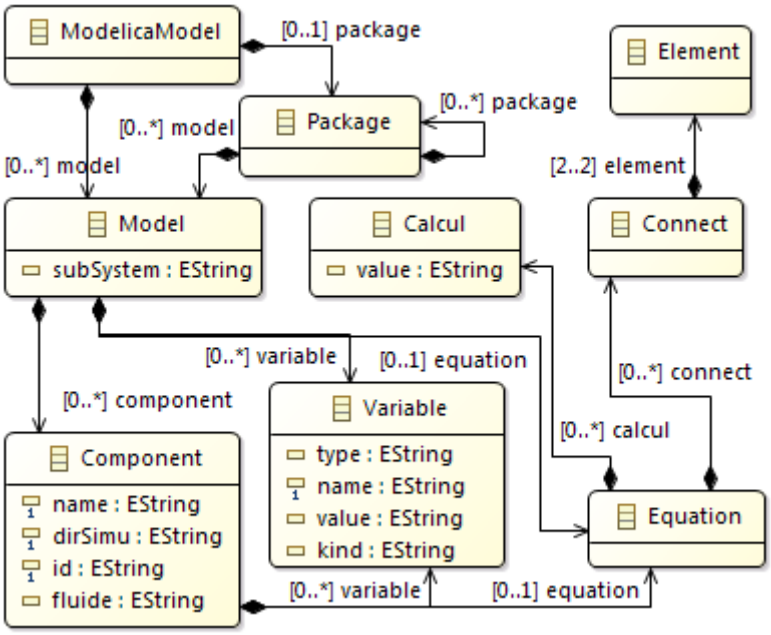


FIGURE 6.7 – Extrait du métamodèle d'un modèle de simulation

Un modèle de simulation (*ModelicaModel* sur la Figure 6.7) est composé d'un "Model" correspondant au modèle qui doit être exécuté durant la simulation.. Le langage *Modelica* étant hierarchique, ce "Model" peut être encapsulé dans un "Package". L'attribut *subSystem*

de “Model” fait référence au nom du sous-système qui est modélisé. Ce “Model” contient des “Component”, qui correspondent aux modèles de simulation des éléments instanciés. Chaque “Component” est référencé par un attribut `id` pour le nom de l'instance et un attribut `name` pour le nom de l'élément. Ces informations sont directement issues de la Nomenclature. Le “Model” peut contenir plusieurs “Variable”. Une “Variable” correspond à une variable de simulation interne ou externe. Le “Model” peut aussi contenir une section “Equation”. La section “Equation” peut contenir deux sortes d'équations qui sont liées au langage *Modelica*. En effet, dans *Modelica*, les connexions de flux physique ou d'information qui sont définis par un *connector*, peuvent se faire en utilisant une équation *connect*. Par conséquent, l'équation “Connect” est composée de deux “Element” qui représentent les deux variables à connecter. L'autre type d'équation qui est une forme classique d'équation correspond à l'équation “Calcul”. Un “Component” peut aussi avoir plusieurs “Variable" qui sont internes à la simulation, et une section “Equation”. De plus, un “Model” ou un “Component” peut avoir des équations initiales.

La première phase de construction du modèle de simulation débute par l'étape d'instanciation (étape 3 du flot). Cette étape permettant d'obtenir le modèle de simulation sans la communication OPC est mise en oeuvre de la façon suivante (Figure 6.8). Les modèles des éléments sont instanciés en premier (étape 3a), puis ensuite ce sont les connexions (étape 3b). Les modèles instanciés ainsi que leurs connexions sont ensuite utilisés pour obtenir le modèle de simulation (sans la communication).

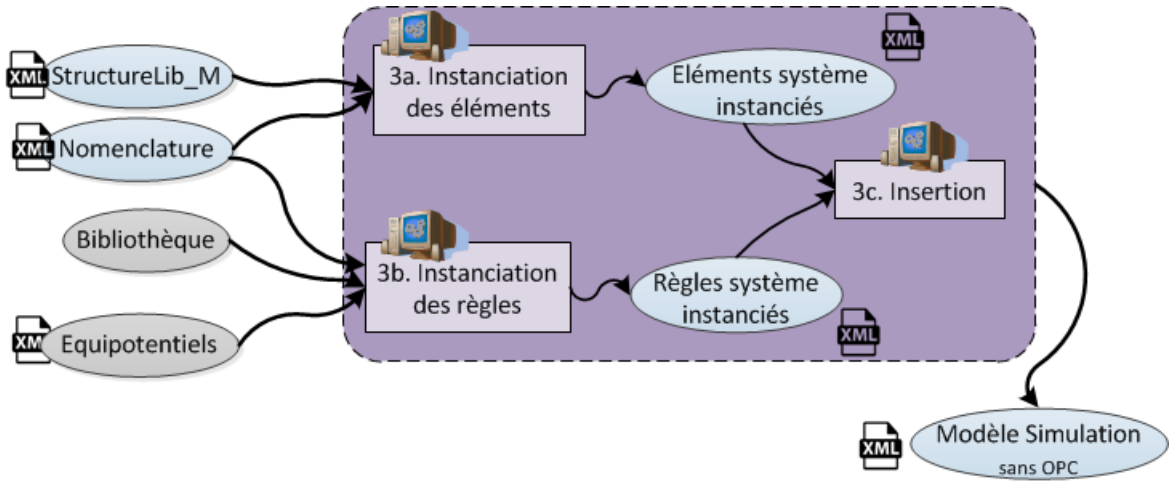


FIGURE 6.8 – Etape 3 : Instanciation des éléments et des connexions

Pour instancier le modèle *Modelica* des éléments qui sont inclus dans la Librairie Modelica générée, nous utilisons le modèle de sa structure (*StructureLib_M*) et la Nomenclature. Résultant de l'étape 3a , le modèle *ElementSyst Instancié* contient un “Model” qui est composé de “Component” (les modèles instanciés des éléments), de plusieurs “ Variable” et d'une section “Equation”. Ces “Variable” correspondent aux variables internes d'entrée/sortie de la simulation. La section “Equation” contient les équations reliant ces variables à celles du “Component” considéré.

Ensuite, pour créer les connexions entre les modèles des éléments (étape 3b), les règles de connexion liées à l'instrumentation sont instanciées, ainsi que celles correspondant aux flux physique. Pour obtenir les règles de connexion des flux informationnels (par exemple lié à l'instrumentation), les règles génériques contenues dans les vues de simulation de la Bibliothèque sont collectées. Ces règles génériques sont définies par le métamodèle des règles (Figure 6.9). Une "Regle" est composée d'une "Reference", dont l'attribut `name` fait référence à l'élément auquel appartient la règle de connexion, et d'un ou plusieurs "ConnectTo" précisant le genre de la variable à connecter (attribut `kind`) et à quel élément (attribut `nature` du ou des "Ci" qu'il contient). Le modèle des règles génériques d'un élément (*Regles* sur la Figure 6.9) est ainsi composée d'une ou plusieurs "Regle".

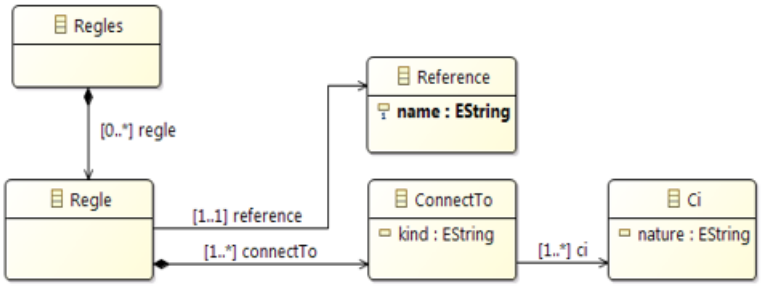


FIGURE 6.9 – Le métamodèle des règles

Les règles génériques collectées sont ensuite contextualisées en utilisant la Nomenclature, puis transformées en équation de type *connect* afin d'être conforme au métamodèle d'un modèle de simulation (cf Figure 6.7). L'obtention des règles de connexion des flux physiques, se fait quant à elle en utilisant le modèle des Equipotentiels. Suite à l'étape 3b, on obtient ainsi le modèle *RèglesSyst Instanciées* dont la section *Equation* contient toutes les connexions liées aux flux informationnels et physiques. Les modèles intermédiaires résultant (*ElementSyst Instancié* et *RèglesSyst Instanciées*) sont combinés dans l'étape 3c, pour générer les modèles de simulation du procédé, à savoir le *Modèle de Simulation sans OPC*.

L'étape suivante se focalise ainsi sur l'ajout de la communication OPC pour permettre les échanges avec le système de contrôle/commande. Les variables OPC requises sont ainsi instanciées à partir de la Nomenclature (étape 4). Ces variables sont ensuite intégrées dans le modèle de simulation sans communication OPC, pour obtenir le modèle de simulation complet (*Modèle de Simulation Complet*). Toutefois ce modèle est encore un modèle intermédiaire : ce n'est pas du code *Modelica*, mais un fichier XML. La dernière étape du flot de génération (étape 5) consiste donc à transformer le *Modèle de Simulation Complet* en code *Modelica*. Nous générons ainsi le modèle de simulation final, *ModeleSimu.mo*, dans le langage *Modelica*.

A travers cette proposition que nous venons de présenter, un effort de formalisation a été fait, pour générer automatiquement un modèle de simulation Modelica du procédé, à partir du schéma P&ID du procédé. Et ce, afin de proposer une démarche qui puisse être utilisée dans un contexte industriel, sans dépendre d'un logiciel propriétaire particulier. Après importation du modèle de simulation généré dans un logiciel Modelica et de la librairie *Modelica* associée

(*Lib_Modelica.mo*), la simulation peut être exécutée afin de vérifier le bon fonctionnement du système de conduite. Toutefois une étape d'initialisation du modèle est nécessaire selon le scénario à exécuter.

6.4 Bilan/Synthèse

Pour éviter la découverte tardive d'erreurs de conception, un intérêt est porté sur la mise en oeuvre de tests des systèmes de conduite de procédés en parallèle des activités de conception. Par conséquent, le comportement du procédé doit être simulé. Néanmoins, l'obtention des modèles de simulation demande un effort de modélisation conséquent, un savoir faire en simulation, et l'obtention manuelle des modèles de simulation est chronophage.

A cet égard, nous souhaitons faciliter l'utilisation de vérification par simulation dans l'industrie, par la mise en place d'une approche de génération automatisée de modèles de simulation. Nous proposons ainsi, un flot de génération automatisé de modèle de simulation afin de tester les systèmes de conduite pour des procédés de type gestion de fluide. Afin de tirer pleinement avantage de la génération automatique, notre proposition est basée sur un flot de conception automatisé de systèmes de contrôle/commande.

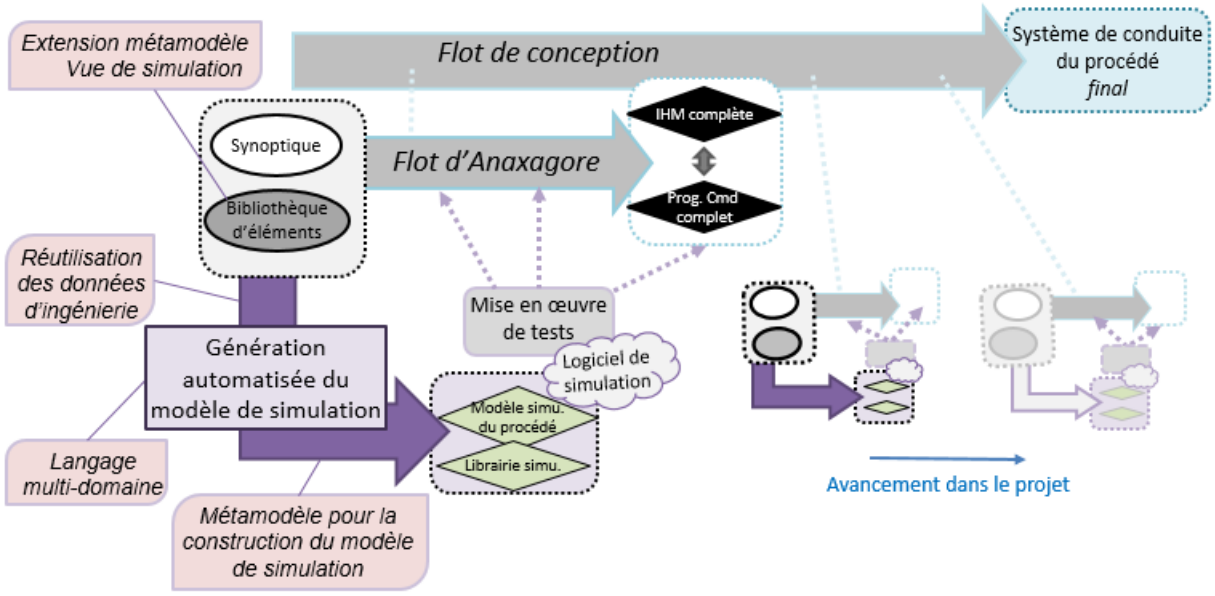


FIGURE 6.10 – Apport de notre contribution pour faciliter l'obtention de la simulation

A partir du P&ID et d'une Bibliothèque basée sur le concept de *vue*, nous générons le modèle de simulation et la libraire de simulation associée, dans le langage cible *Modelica* (Figure 6.10). Ces modèles d'entrée proviennent de [Bignon et al., 2013], néanmoins le métamodèle de la Bibliothèque a été étendu pour nos besoins. D'autre part, la première des cinq étapes constituant notre flot de génération de modèles de simulation est également issue des travaux de [Bignon et al., 2013]. A partir de cette première étape, une librairie de simulation, en code

Modelica, est générée. Concernant la phase de génération du modèle de simulation, nous avons défini un métamodèle commun. Ainsi tous les modèles générés, qu'ils soient intermédiaires ou non, sont conformes au même métamodèle. La construction du modèle consiste à instancier les modèles des éléments ainsi que leurs connexions, puis à instancier la communication OPC pour les échanges avec la partie de commande. Enfin, la dernière étape permet de traduire le modèle de simulation complet en code *Modelica*. La Figure 6.10 synthétise les apports de notre contribution, qui a par ailleurs fait l'objet d'une publication [Prat et al., 2017].

Un des objectifs visés au travers de ces travaux de thèse consiste à faciliter l'obtention des modèles de simulation en parallèle des activités de conception, pour limiter la détection tardive d'erreur. Au travers de cette contribution, un effort de formalisation a été réalisé pour permettre la mise en oeuvre d'une approche de génération automatisée des modèles de simulation, et ce dans un contexte industriel. De plus, afin d'atteindre un deuxième objectif qui vise à introduire une démarche de vérification par simulation dans un flot de conception automatisé, nous avons proposé une démarche de génération automatisée qui peut s'intégrer dans un flot de conception automatisé (levant ainsi le troisième verrou identifié). L'intégration de notre proposition dans un flot de conception automatisé, comme celui d'Anaxagore, offre l'avantage de permettre le test du système de conduite du procédé dès les premières phases de conception. Ces tests peuvent concerner tant la partie de contrôle-commande que l'IHM de supervision. D'autre part, les tests peuvent également être introduits dans le flot de conception automatisé, comme illustré sur la Figure 6.11.

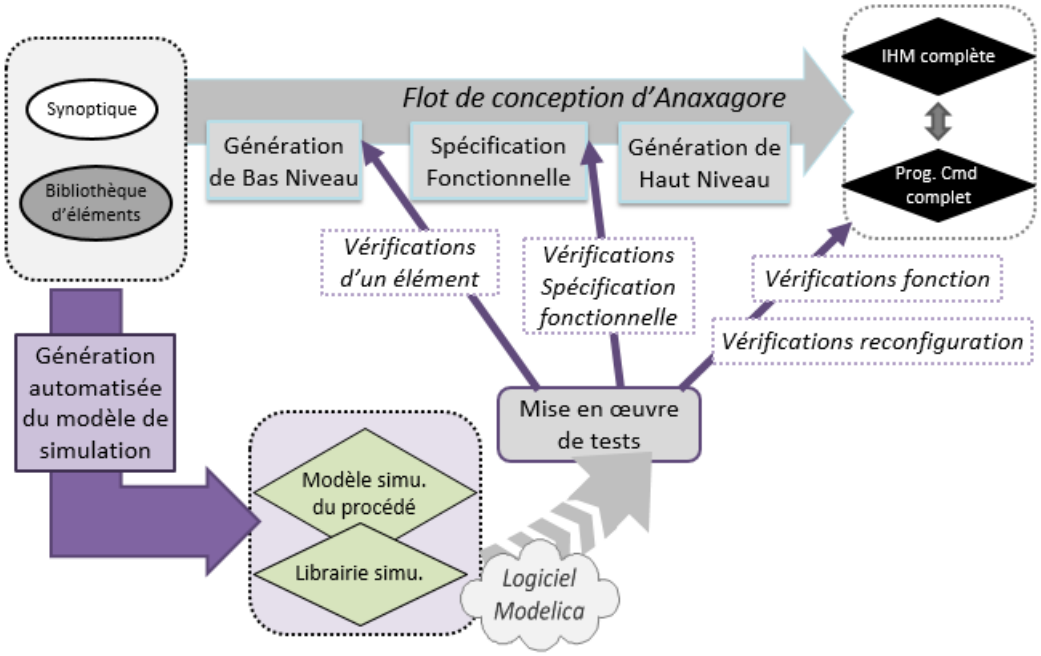


FIGURE 6.11 – Démarche de vérification intégrée au flot de conception d'Anaxagore

La prochaine partie de ce manuscrit sera ainsi consacrée à l'application de nos propositions théoriques, apportant à la fois une preuve de concept et une preuve d'usage.

Partie C -

Applications

Cette troisième et dernière partie de ce manuscrit concerne la mise en oeuvre de nos propositions théoriques par rapport à nos objectifs. Ainsi, il s'agira d'appliquer dans un premier temps notre contribution visant à faciliter l'obtention d'un prototype virtuel d'un procédé. Puis, dans un second temps, il s'agira de mettre en oeuvre une démarche de vérification au plus tôt dans le flot de conception d'Anaxagore afin de mettre à l'épreuve l'ensemble de nos contributions.

Le Chapitre 7 concerne l'implémentation du *flot de simulation*. Il s'agit notamment d'appliquer la proposition de génération automatique de modèles de simulation afin d'apporter une preuve de concept.

Le Chapitre 8 concerne la démarche de vérification. Il illustre l'application de nos propositions et permet ainsi d'obtenir une preuve d'usage.

7

Preuve de concept : Implémentation du "flot de simulation"

Sommaire

Dans les précédents chapitres des propositions théoriques ont été faites afin de faciliter l'obtention de la simulation. Dans ce chapitre, nous présentons leurs implémentations, puis une application sur un cas, afin d'apporter une preuve de concept.

Pour rappel, le flot de simulation proposé est constitué de cinq étapes, comme illustré sur la Figure 7.1, et est basé sur le flot de conception d'Anaxagore. La génération des modèles de simulation se fait en réutilisant des modèles issus du flot de conception (le Synoptique et le modèle des Equipotentiels). De plus, la Bibliothèque basée sur le concept de *vue*, qui est utilisée dans le flot de conception, a été étendue pour pouvoir être utilisée dans le « flot de simulation ».

Avant de détailler l'implémentation de chaque étape du flot de simulation, nous nous intéressons en premier lieu aux modèles d'entrée du flot, objet de la section qui suit.

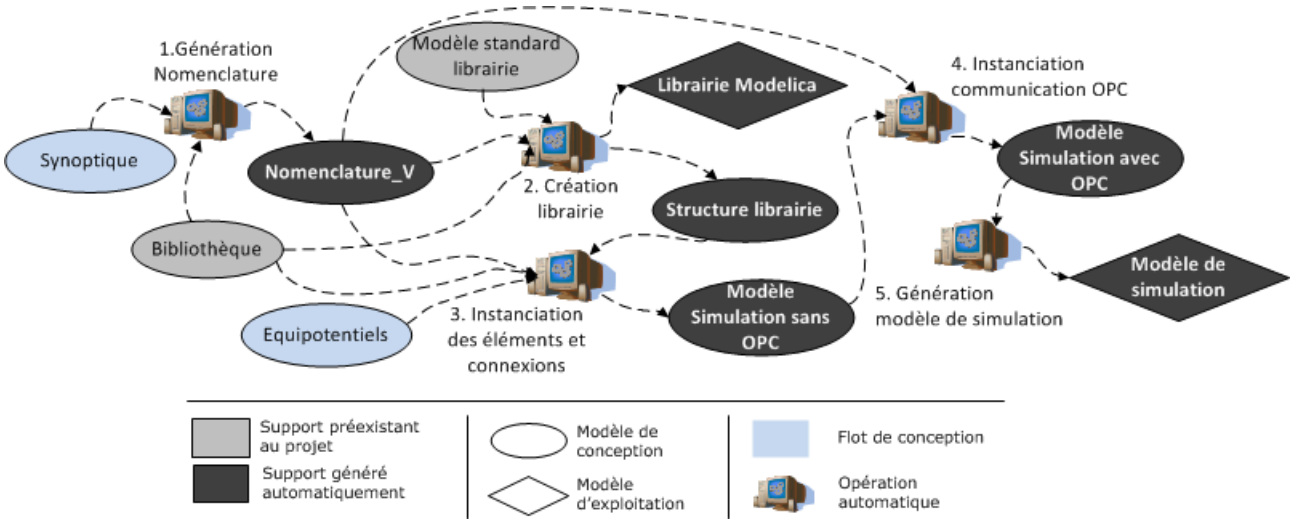


FIGURE 7.1 – Flot de génération des modèles de simulation

7.1 Les modèles d'entrée du *flot de simulation*

Dans cette section nous présentons rapidement le Synoptique et le modèle des Equipotentiels qui sont directement issus du flot de conception d'Anaxagore. Puis, nous détaillerons les modifications apportées à la Bibliothèque d'éléments d'Anaxogore, dans le cadre de notre *flot de simulation*.

7.1.1 Le Synoptique et le modèle des Equipotentiels

Le **modèle Synoptique** est une représentation informatique du schéma P&ID (diagramme Synoptique sur la figure 7.2). Ce modèle contient toutes les informations liés aux symboles (`shape`) et aux connexions (`bond`) présentes sur le diagramme. De façon générale, les symboles représentent les éléments du procédé [1].

Chaque élément est représenté sur le Synoptique par un symbole dédié et un identifiant, comme illustré sur la Figure 7.2, qu'il s'agisse du diagramme ou du fichier xml. A chaque élément présent dans le diagramme correspond une balise `<shape>` dans le modèle Synoptique. L'attribut `name` fait référence au type d'élément (par exemple, V2VM) et l'attribut `id` correspond à l'identifiant (par exemple, V2VM01). Cet identifiant peut aussi être considéré comme le nom de l'instance de l'élément, V2VM01 et V2VM02 sont alors des instances de l'élément V2VM.

Les liens entre les éléments se font à partir de leurs ports, afin de modéliser les échanges de flux d'informations ou de flux physiques. C'est donc via ces ports que les éléments (qu'ils soient de nature matérielle ou logicielle) peuvent être connectés. Les différents ports dont dispose un élément sont définis par la balise `<port>`. La vanne V2VM dispose donc de trois

1. Certains symboles correspondent, non pas à des éléments du procédé, mais à des éléments de supervision.

ports (Figure 7.2). Les ports 1 et 2 de V2VM01 (c'est à dire ceux dont l'attribut `number` vaut "1" et "2") représentent les points de connexion permettant au fluide d'entrer ou sortir de la vanne. Le port 1 du transmetteur de niveau LT0002 représente le signal de mesure du capteur tandis que le port 2 représente le signal d'information sur la mesure.

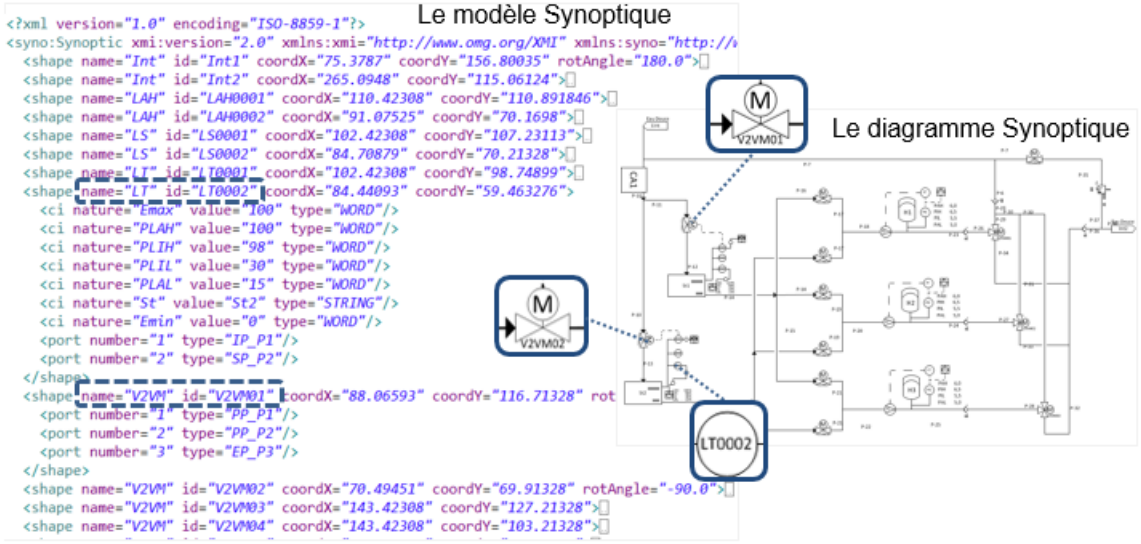


FIGURE 7.2 – Exemple de Synoptique

Tous les liens entre les éléments dans le diagramme sont représentés par des connexions dans le modèle (Figure 7.3). A chaque connexion correspond une balise `<bond>` qui contient au moins deux extrémités (`extremity`). Ces extrémités correspondent aux ports des éléments. Un exemple de connexion entre un transmetteur de niveau et une soute est donné sur la Figure 7.3. La première extrémité de la connexion correspond au port 1 de LT0002. La seconde extrémité correspond au port 3 de la soute St2. Ainsi, cette connexion modélise la mesure du niveau de la soute St2 par le capteur LT0002.

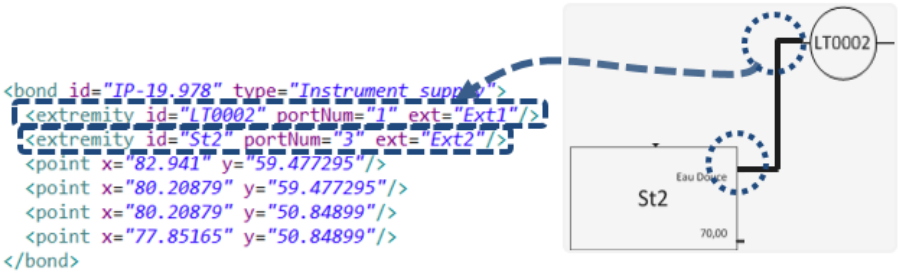


FIGURE 7.3 – Exemple d'une connexion sur le Synoptique

Le **modèle des Equipotentiels** représente les points de connexion entre les éléments où le potentiel (du fluide) est le même. Un *équipotentiel* est une représentation de plusieurs connexions, modélisant la circulation du fluide, qui sont connectées entre elles. Un exemple d'équipotentiel est illustré sur la Figure 7.4. Il correspond au point de connexion entre les vannes V2VM03 et V2VM04 et le groupe hydrophore H1. Le modèle des Equipotentiels est ainsi une liste de tous les équipotentiels présent sur le Synoptique.

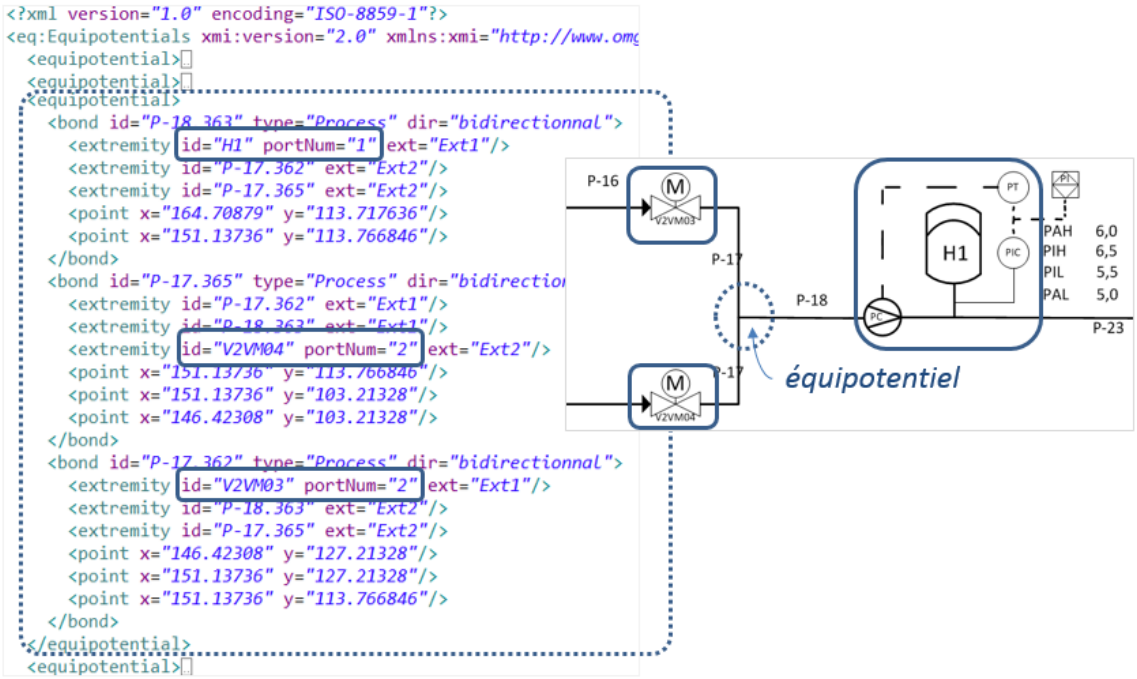


FIGURE 7.4 – Exemple d'un modèle des Equipotentiels

7.1.2 La Bibliothèque d'éléments

Comme proposé au chapitre 6, nous avons étendu le méta-modèle de la bibliothèque d'éléments standards du flot de conception d'Anaxagore. Le **modèle de la Bibliothèque** a donc été modifié en conséquence comme illustré sur la Figure 7.5.

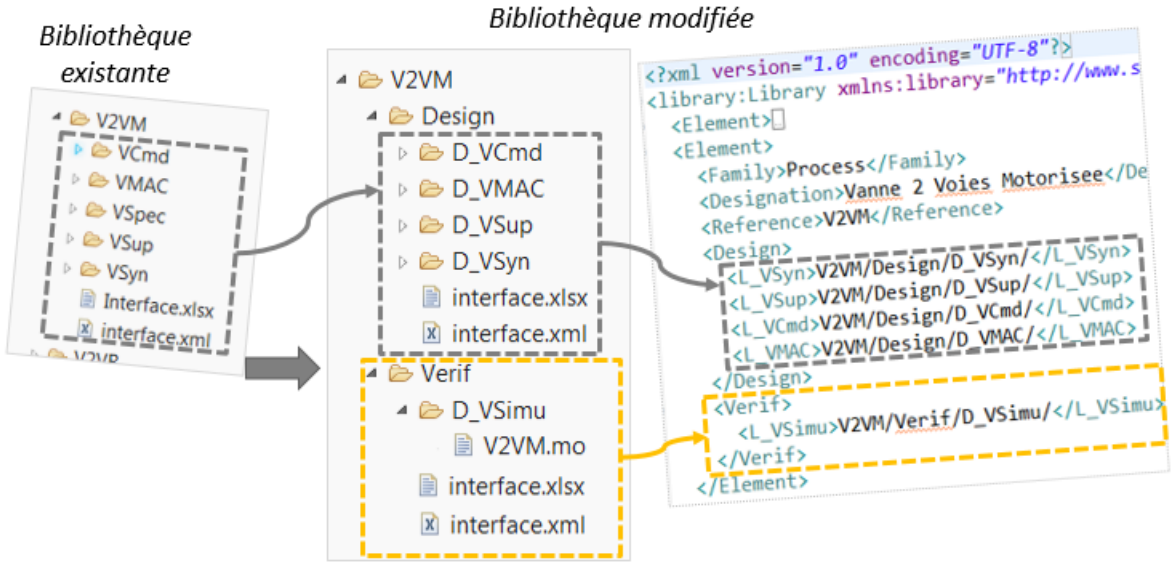
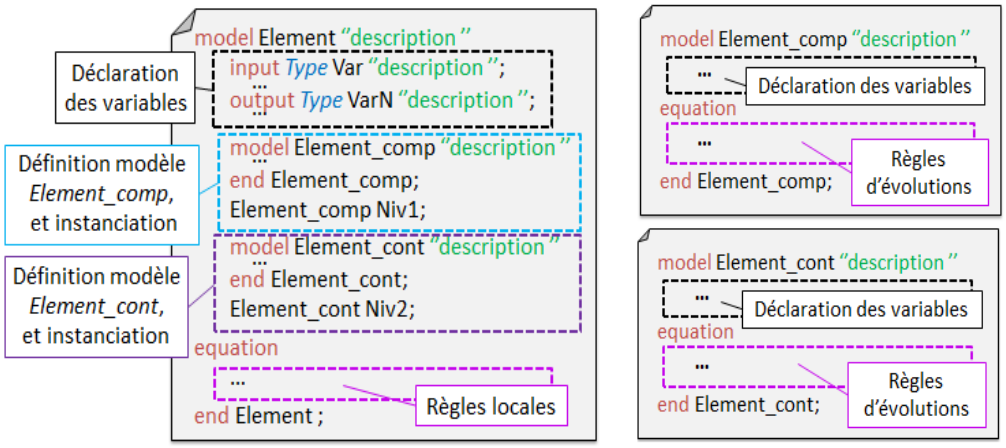


FIGURE 7.5 – Modification de la structure de la Bibliothèque Standard d'éléments

Outre l'ajout d'une vue de simulation, nous avons regroupé les *vues* associées au flot de conception dans un dossier *Design* et les *vues* associées à la vérification dans un dossier *Verif*,

permettant ainsi de séparer la conception de la vérification. Le dossier *Verif* contient la vue de simulation de l'élément, et le fichier *interface.xml* qui liste les interfaces du modèle de simulation. Les conventions de nommage des *vues* ont également étés modifiées. Les dossiers des différentes *vues* commencent par « D_ », tandis que les attributs faisant référence au chemin des *vues* sont désignés par « L_ ». Le modèle de simulation (en langage Modelica) d'un élément est contenu dans sa vue de simulation, à savoir *D_ VSimu*. Par convention, le fichier Modelica est nommé d'après le nom de l'élément suivi de l'extension *.mo*. Il s'agit, dans le cas d'une vanne à deux voies motorisée (V2VM sur la Figure 7.5) du fichier *V2VM.mo*.

Le modèle de simulation d'un élément est construit en suivant la proposition de structuration des modèles de simulation (cf Chapitre 5). Selon la nature de l'élément, le modèle comporte un Niveau Composant et/ou un Niveau Contextualisation. Le modèle de simulation d'un élément a une structure hiérarchisée qui inclut les Niveaux Composant et Contextualisation à travers deux sous-modèles (Figure 7.6). Le Niveau Règles (règles locales sur la Figure 7.6a), étant d'un plus haut niveau d'abstraction, permet entre autres de faire le lien entre les sous-modèles.



(a) Structure du modèle (b) Structure des sous-modèles

FIGURE 7.6 – Aperçu de structuration du modèle d'un élément en langage Modelica

Afin de pouvoir créer les liens entre les variables échangées, associées aux modèles de simulation des éléments, le **modèle des interfaces** de chacun d'entre eux est établi. Toutes les interfaces d'un modèle de simulation sont ainsi définies dans le fichier *interface.xml* situé dans le dossier *Verif* de l'élément(cf Figure 7.5). Les variables d'entrée/sortie (interfaces) des modèles de simulation peuvent être de nature très différentes. L'attribut **kind** d'une interface de simulation peut ainsi prendre différentes valeurs :

- InPhy/OutPhy : pour les variables en provenance/à destination de la partie commande ;
- Alea : pour les variables permettant d'introduire des défauts ;
- OutputS : pour des variables de sortie de la simulation (observables) ;
- InputS : pour des variables d'entrée internes à la simulation (autres que les aléas) ;
- MesPhy : pour les variables représentant des mesures physiques ;
- PhyFlow : pour les variables représentant des flux physiques bidirectionnels ;

- InFlow/OutFlow : pour les variables représentant des flux physiques orientés.

Un exemple d'interface de simulation pour l'élément V2VM est donné Figure 7.7. Les interfaces *CmdO* et *FdcO* correspondent à des variables échangées avec la partie commande. Ce type de variable est associé à une modélisation Niveau Composant. Les interfaces *port1* et *port2*, au contraire, sont associées à une modélisation Niveau Contextualisation, elles permettent de représenter la circulation du fluide.

Reference	Information	Designation	Type	Kind	View
V2VM	CmdO	Commande d'ouverture vers la vanne	BOOL	InPhy	L_VSimu
V2VM	ffdcO	Defaut capteur fin de course ouverture	INT	Alea	L_VSimu
V2VM	FdcO	Fin de course d'ouverture	BOOL	OutPhy	L_VSimu
V2VM	opened	Position vanne ouverte	BOOL	OutputS	L_VSimu
V2VM	port1	Flux physique entrant dans la vanne	REAL	PhyFlow	L_VSimu
V2VM	port2	Flux physique sortant de la vanne	REAL	PhyFlow	L_VSimu

FIGURE 7.7 – Extrait des interfaces de simulation de V2VM

D'autre part, certains éléments, en particulier ceux d'instrumentations, disposent d'un modèle supplémentaire : un **modèle de règles**. Il s'agit de règles spécifiant, selon la nature d'une interface, à quel élément elle peut être connectée. Un capteur de mesure peut potentiellement être associé à différents éléments. Pour savoir à quel type d'éléments le capteur peut être connecté, un fichier *regles.xml* est défini (dans la vue de simulation). La Figure 7.8 montre le modèle de règles d'un transmetteur de niveau (LT). Ce modèle contient une seule règle qui stipule que l'interface de simulation représentant la mesure du capteur (*kind="MesPhy"*) doit être connectée à une interface d'une soute (St) dont *kind="MesPhy"*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:Regles xmlns:xmi="http://www.segula.fr/simulation/regles">
    <regle>
        <reference name="LT"/>
        <connectTo kind="MesPhy">
            <ci nature="St"/>
        </connectTo>
    </regle>
</xmi:Regles>
```

FIGURE 7.8 – Le fichier regle.xml de l'élément LT

Maintenant que les modèles d'entrée du flot de simulation sont présentés, intéressons-nous à l'implémentation de chacune des étapes de celui-ci (cf Figure 7.1), objets des sections suivantes.

7.2 Etape 1 : Génération de la Nomenclature

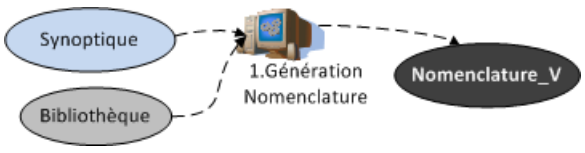


FIGURE 7.9 – Etape 1

La génération de la Nomenclature du *flot de simulation* est similaire à celle définie dans le cadre du *flot de conception Anaxagore*. Afin d'éviter toute confusion avec la Nomenclature du flot de conception, celle du flot simulation est qualifiée de *Nomenclature_ V* (Figure 7.9). La Nomenclature_V est une synthèse des informations présentes dans le modèle Synoptique enrichies par les interfaces de chaque élément, que ce soit par rapport à la vue de simulation ou à la vue de commande. La formalisation de cette première étape du flot, synthétisée sur la Figure 7.10, est similiaire à celle présentée dans les travaux de [Bignon, 2012].

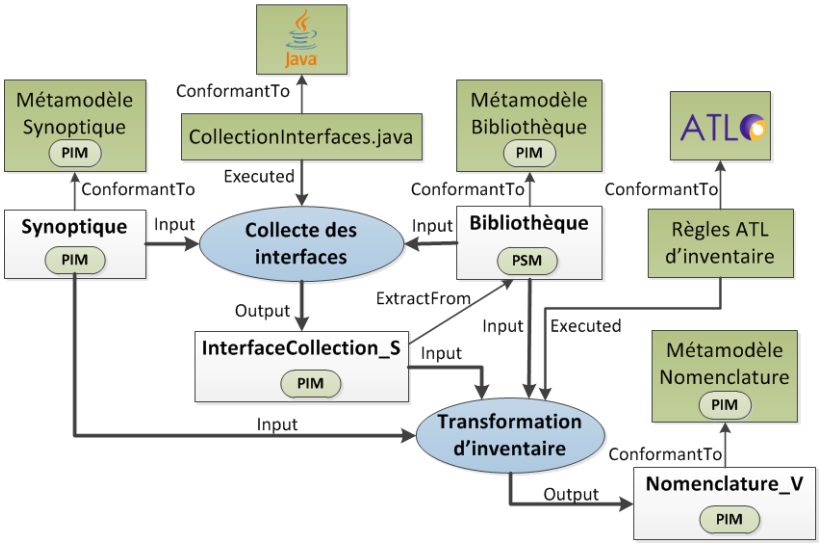


FIGURE 7.10 – Formalisation de la génération de la Nomenclature_V

A partir du modèle Synoptique (conforme à son métamodèle) et du modèle de la Bibliothèque (conforme à son métamodèle), le programme **CollectionInterfaces.java** génère le modèle des interfaces qui est conforme au métamodèle des interfaces. Ce modèle des interfaces, *InterfaceCollection_ S*, contient uniquement les interfaces associées aux vues de commande et de simulation. Pour chaque type d'élément présent dans le Synoptique, le modèle de la Bibliothèque est utilisé pour connaître les emplacements des modèles des interfaces des éléments. Les interfaces associées à la vue de commande sont récupérées à partir du fichier *interface.xml* contenu dans le dossier *Design* de l'élément. Les interfaces liées à la vue de simulation sont quant à elles récupérées dans le fichier *interface.xml* du dossier *Verif*. Les interfaces collectées de chaque élément sont ensuite placées dans le fichier *InterfaceCollection_ S.xml*.

La Nomenclature_V est ensuite générée par l'exécution de la **Transformation d'inventaire**[2] afin d'obtenir une synthèse des informations liées à chaque instance d'éléments du procédé. Les informations contenues dans le Synoptique sont ainsi enrichies par celles du modèle des interfaces et du modèle de la Bibliothèque. Pour chaque `shape` du Synoptique, il est créé une `instance` dans la Nomenclature_V avec la référence de l'élément (par exemple V2VM sur la Figure 7.11) et son identifiant (V2VM01). Chaque instance contient également les attributs `L_VSimu` et `L_VCmd`, récupérés à partir du modèle de la Bibliothèque. A partir du

2. Cette Transformation d'inventaire a été définie dans le cadre du flot de conception d'Anaxagore.

modèle InterfacesCollection_S, les interfaces des éléments sont instanciées et rajoutées dans la Nomenclature_V.

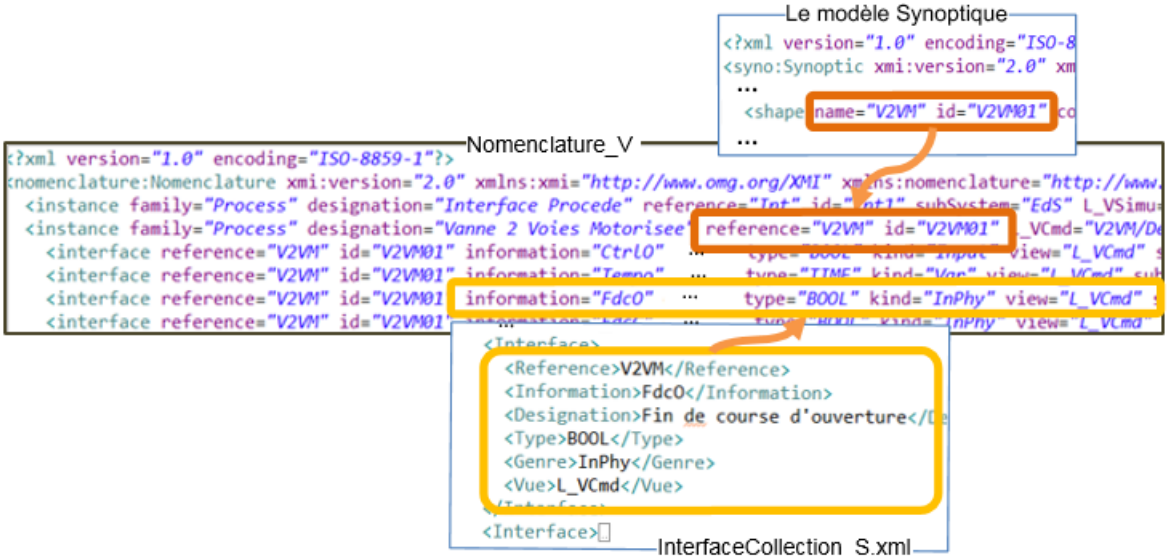


FIGURE 7.11 – Extrait de la Nomenclature_V

L'instance V2VM01 de la Nomenclature_V (Figure 7.11) contient des balises `<interface>` correspondant aux interfaces collectées associées à V2VM (attribut `reference`="V2VM"). De plus, toutes les connexions (balises `<bond>`) présentes dans le Synoptique sont récupérées et insérées dans la Nomenclature _V. Ce modèle regroupe ainsi toutes les informations liées aux instances des éléments du système, et de leurs connexions.

7.3 Etape2 : Génération de la librairie de simulation

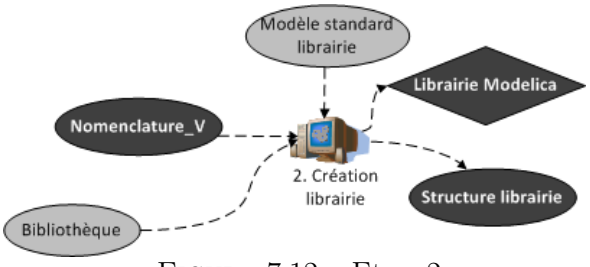


FIGURE 7.12 – Etape2

Le but de la deuxième étape du flot de simulation est de générer la librairie spécifique de simulation contenant les modèles de simulation élémentaires qui seront utilisés pour construire celui du procédé et l'exécuter.

La formalisation de l'implémentation de cette étape est représentée sur la Figure 7.13. Suivant le principe évoqué au chapitre 6, deux modèles sont générés. Il s'agit d'une part du modèle *Struct_lib* représentant la structure de la librairie, utilisé pour la construction du modèle de simulation du procédé. Et d'autre part, de la librairie en langage Modelica, *Lib_Modelica*, utilisée pour exécuter le modèle de simulation du procédé.

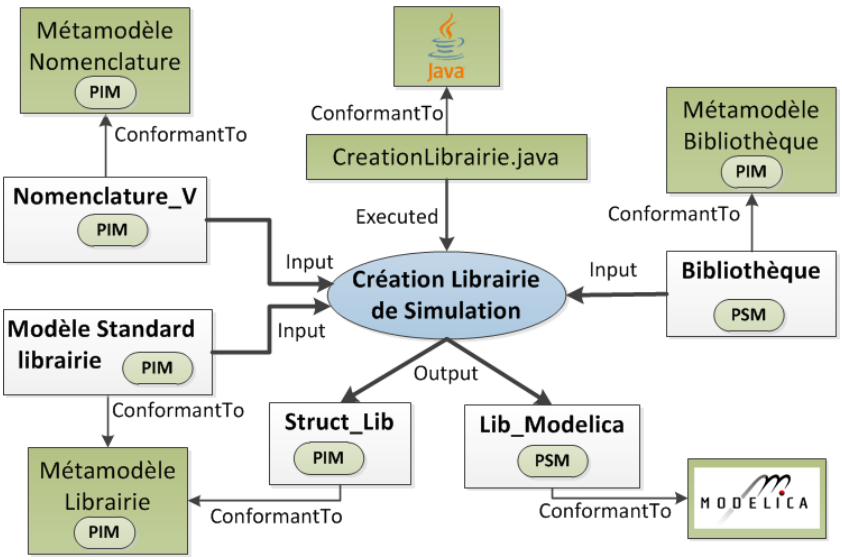


FIGURE 7.13 – Formalisation de la Génération de la librairie de simulation

A partir de la Nomenclature_V et du Modèle Standard de la librairie (conforme au métamodèle de la Libraire), le programme **CreationLibrairie.java** génère le modèle de la structure de la librairie (également conforme au métamodèle de la Librairie). La liste des éléments dont il faut récupérer le modèle de simulation est extraite de la Nomenclature_V. Pour chaque élément, sont récupérés les attributs `family`, `reference` et `L_VSimu`. La valeur de l'attribut de `family` permet de savoir où le modèle de simulation de l'élément doit être placé dans la librairie. Le Modèle Standard de la librairie (qui est le squelette "vide" de la librairie) est ensuite complété afin d'obtenir la structure de la librairie spécifique (Figure 7.14). Pour ce faire, les informations liées au modèle de simulation sont encapsulées dans une balise `<component>` dont la valeur de l'attribut `name` correspond au nom du modèle de simulation, et celle de l'attribut `dirSimu` correspond à l'emplacement du modèle de simulation dans la Bibliothèque d'élément Standard.

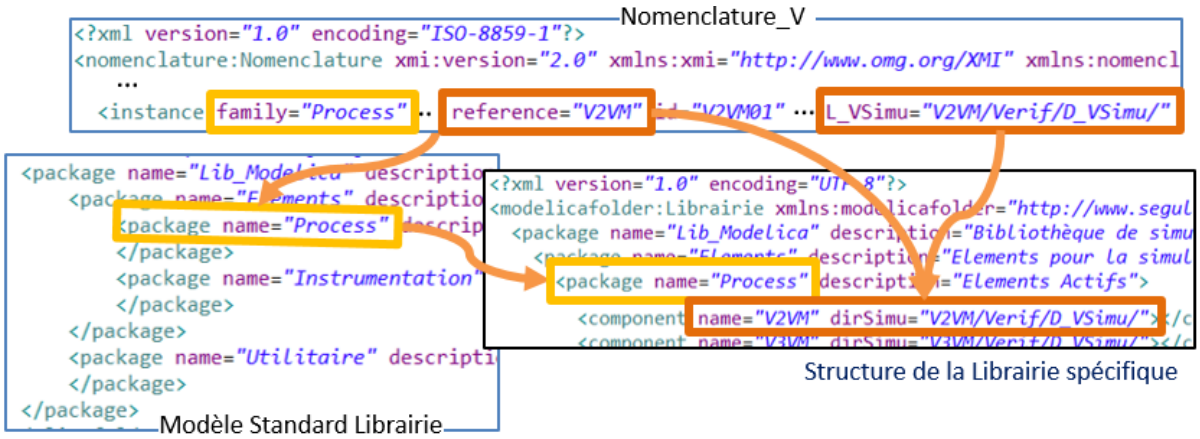


FIGURE 7.14 – Obtention de la structure de la librairie

Comme illustré sur la Figure 7.14, une vanne à deux voies motorisée (V2VM) appartient à la famille *Process*, son modèle de simulation sera donc positionné dans le `package` *"Process"*

de la librairie. Le modèle de la structure de la librairie contient ainsi une balise `<component>` dont l'attribut `name` vaut *"V2VM"*, dont le parent est la balise `<package>` ayant l'attribut `name` de valeur *"Process"*. D'autre part, la valeur de l'attribut `dirSimu` de ce `<component>` correspond à l'emplacement du dossier de la vue de simulation de V2VM dans la Bibliothèque.

Ce modèle contient ainsi toutes les informations nécessaires pour récupérer les modèles de simulation des éléments que doit contenir la librairie de simulation. La construction de la librairie s'appuie donc sur celui-ci. A partir des valeur des attributs `dirSimu`, le contenu des modèles de simulation correspondant est récupéré et rajouté au modèle afin d'obtenir un modèle de la librairie au format xml (Figure 7.15). Le code source de chaque modèle de simulation est ainsi inséré entre les balises `<composant>` et `</composant>` concernées du modèle de la librairie. Par exemple, pour le modèle de simulation de V2VM (Figure 7.15), le code source contenu dans la vue de simulation est inséré dans la balise `<component>` dont `name`="V2VM". Le programme **CreationLibrairie.java** traduit ensuite ce modèle de la librairie en code Modelica (en parsant le fichier xml).

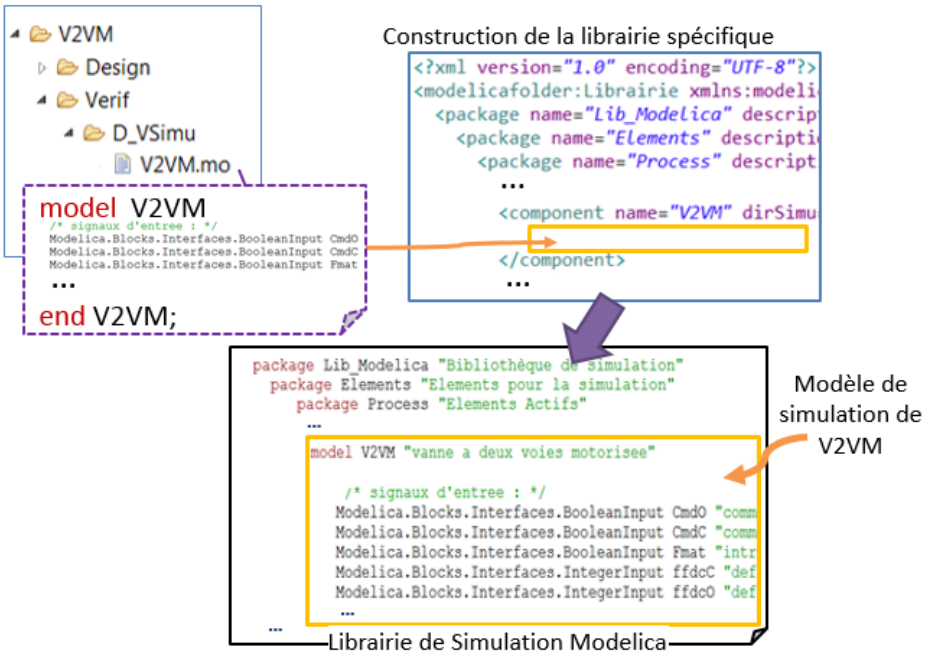


FIGURE 7.15 – Obtention de la librairie de simulation

La librairie de simulation Modelica générée (Figure 7.15) contient ainsi les modèles de simulation des éléments (le code source) qui seront par la suite instanciés afin d'obtenir un modèle de simulation du procédé.

7.4 Etape3 : Instanciation des modèles et connexions

Le but de cette étape est d'obtenir le modèle de simulation du procédé, sans la communication OPC (Figure 7.16). Comme évoqué chapitre 6, il s'agit d'instancier les modèles de simulation des éléments (étape 3a), d'instancier les règles (étape 3b), puis de combiner les

modèles résultants (étape 3c) pour obtenir le modèle de simulation sans la communication OPC.

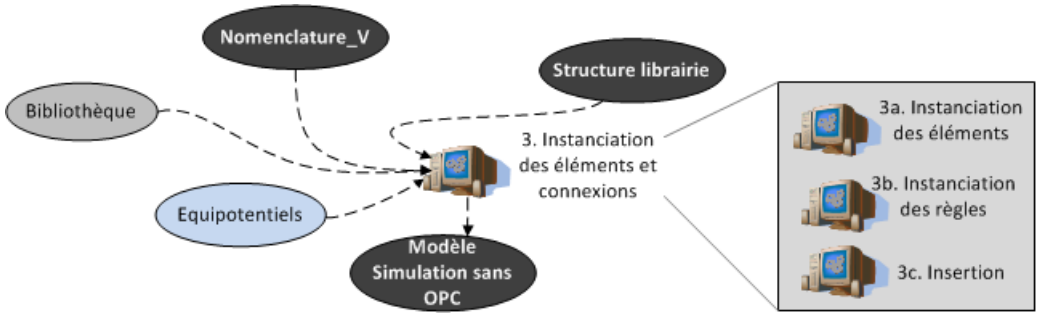


FIGURE 7.16 – Etape 3

7.4.1 Etape 3a. Instanciation des modèles de simulation

L'**étape 3a**, consistant à instancier les modèles de simulation des éléments, est formalisée sur la Figure 7.17. Outre l'instanciation des modèles de simulation de la Librairie de Simulation par rapport aux instances présentes dans la Nomenclature_V, il s'agit également dans cette étape d'instancier les variables d'entrée/sortie internes à la simulation.

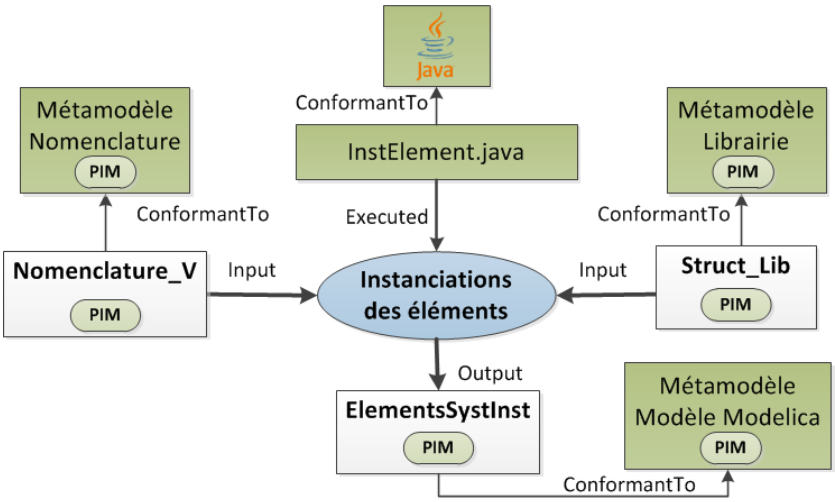


FIGURE 7.17 – Formalisation de l'étape 3a

A partir du modèle Nomenclature_V (qui est conforme au métamodèle de la Nomenclature) et du modèle Struct_Lib (qui est conforme au métamodèle de la Librairie), l'exécution du programme **InstElement.java** génère le modèle ElementsSystInst conformément au métamodèle d'un Modèle Modelica.

Le programme récupère tout d'abord les composants présents dans le fichier Struct_lib.xml (correspondant aux modèles de simulations à instancier). La phase d'instanciation, à proprement parler, débute après que la structure du fichier ElementsSystInst.xml ait été créée. A partir des instances de la Nomenclature_V, les composants correspondant dans le modèle Struct_Lib sont récupérés pour créer les instances des composants dans le modèle. Par exemple, pour créer l'instance V2VM01 dans le modèle de simulation, le modèle de simulation de V2VM doit être instancié. L'attribut `reference` de l'instance dans la Nomenclature_V

permet de sélectionner le composant de la librairie qui doit être instancié (celui dont l'attribut `name` vaut "V2VM"). Suite à l'instanciation (Figure 7.18), il apparait dans le modèle ElementsSystInst une balise `<component>` où `name="V2VM"` et `id="V2VM01"`, ce qui signifie que l'instance V2VM01 du modèle de simulation V2VM a été définie.

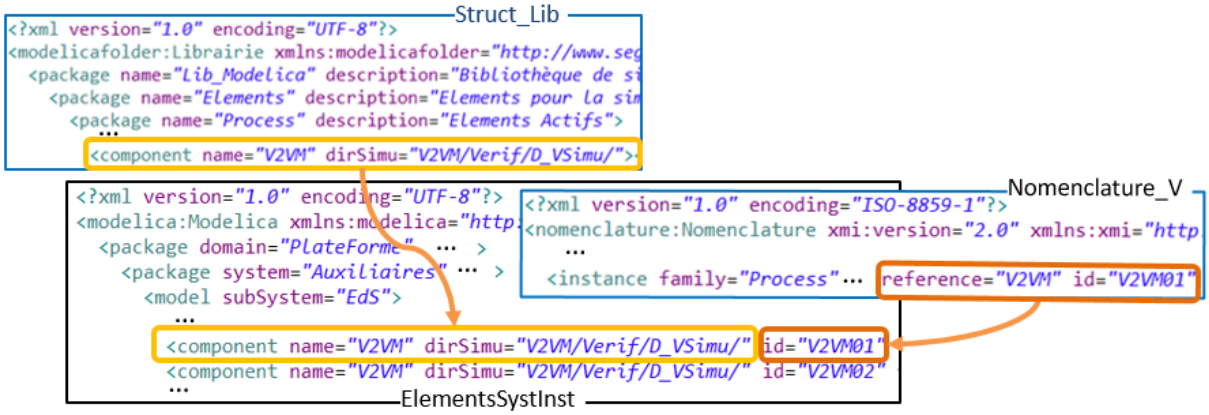


FIGURE 7.18 – Instanciation des modèles des éléments

Puis, les variables internes à la simulation, liées à l'introduction de défauts ou aux sorties de simulation (pour permettre l'analyse des résultats), sont instanciées (Figure 7.19). Pour ce faire, le programme récupère les interfaces des instances de la Nomenclature_V dont l'attribut `kind` a pour valeur *"Alea"* ou *"OutputS"*. Ces variables globales sont instanciées et insérées dans le modèle à l'aide de balises `<variable>` dont l'attribut `name` correspond au nom de la variable. Les valeurs des attributs `type` et `kind` sont récupérées dans celles des `<interfaces>`. Par convention, nous avons choisi de les nommer en reprenant la valeur de l'attribut `information` des interfaces, à laquelle est ajouté le nom de l'instance de l'élément. Par exemple, la variable globale associée à l'interface ffdcO de V2VM01 est appelée ffdcO_V2VM01 dans le modèle de simulation (Figure 7.19).

```
<?xml version="1.0" encoding="UTF-8"?>
<modelica:Modelica xmlns:modelica="http://www.segula.fr/simulation/modelica">
  <package domain="PlateForme">
    <package system="Auxiliaires">
      <model subSystem="EdS">
        <!--- Les instances des modèles de simulation -->
        <component name="V2VM" dirSimu="V2VM/Verif/D_VSimu/" id="V2VM01" family="Process" capacity="-1.0" fluide="EauDouce">
        </component>
        ...
        <!--- Les variables globales instanciees -->
        <variable type="Integer" name="ffdcC_V2VM01" comment="Defaut capteur fin de course fermeture" kind="Alea"></variable>
        <variable type="Integer" name="ffdcO_V2VM01" comment="Defaut capteur fin de course ouverture" kind="Alea"></variable>
        <variable type="Boolean" name="Fmat_V2VM01" comment="Defaut physique de la vanne" kind="Alea"></variable>
        <variable type="Boolean" name="opened_V2VM01" comment="Position vanne ouverte" kind="OutputS"></variable>
        <variable type="Boolean" name="closed_V2VM01" comment="Position vanne fermee" kind="OutputS"></variable>
        ...
        <equation>
            <!--- Les connexions des variables globales avec les variables des modèles -->
            <calcul value="ffdcC_V2VM01 = V2VM01.ffdcC"></calcul>
            <calcul value="ffdcO_V2VM01 = V2VM01.ffdcO"></calcul>
            <calcul value="Fmat_V2VM01 = V2VM01.Fmat"></calcul>
            <calcul value="opened_V2VM01 = V2VM01.opened"></calcul>
            <calcul value="closed_V2VM01 = V2VM01.closed"></calcul>
            ...
        </equation>
      </model>
```

FIGURE 7.19 – Extrait du modèle des éléments instanciés ElementsSystInst.xml

Une fois les variables globales définies, elles doivent être connectées avec les interfaces des

modèles de simulation concernés. Ces connexions sont insérées entre les balises `<equation>` `</equation>`, via la balise `<calcul>`. L'attribut `value` permet de relier la variable globale à celle du modèle de l'élément. La variable globale ffdcO_V2VM01 est ainsi connectée à la variable *ffdcO* de V2VM01 (ffdcO.V2VM01 sur la Figure 7.19).

7.4.2 Etape 3b. Instanciation des connexions

Maintenant que les modèles de simulation des éléments du procédé sont instanciés, il s'agit de générer les connexions entre les modèles (que ce soit par rapport à la circulation du fluide ou par rapport aux mesures effectuées par les capteurs). Ceci est l'objet de l'étape 3b, formalisée sur la Figure 7.20.

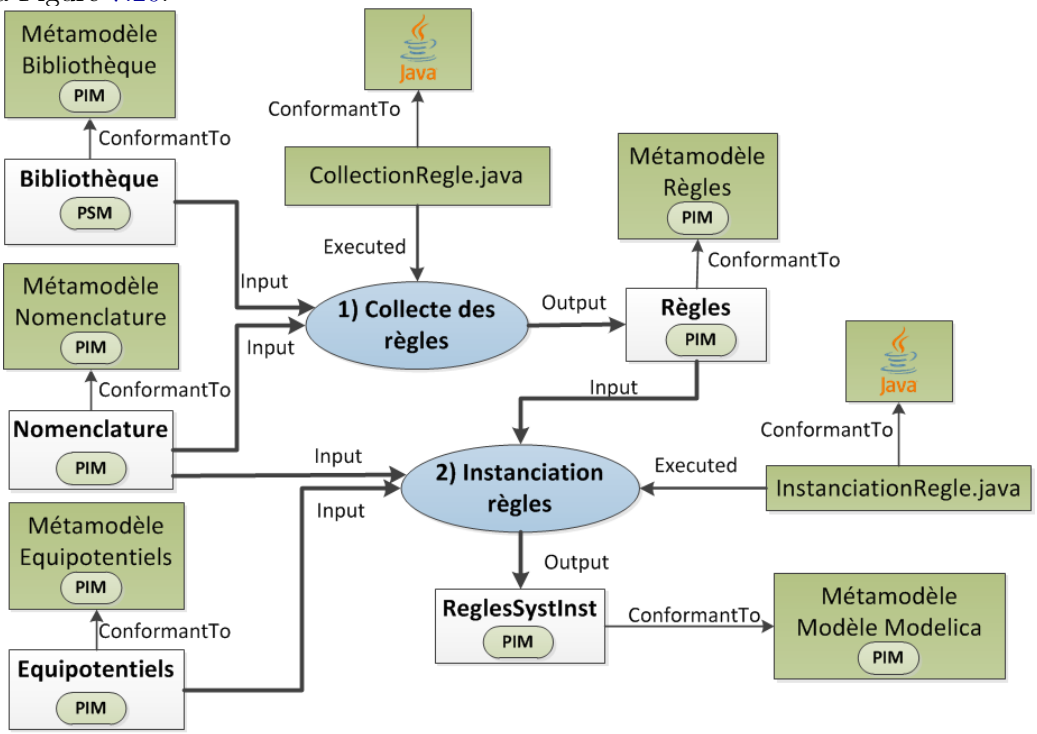


FIGURE 7.20 – Formalisation de l'étape 3b

Dans un premier temps, une collecte des règles contenues dans les *vues* de simulation des éléments (fichier regle.xml) est réalisée par le programme **CollectionRegle.java**. Ainsi, à partir des éléments d'instrumentation présents dans la Nomenclature_V, ce programme collecte les règles contenues dans la Bibliothèque (les modèles de règles des éléments) et génère le modèle des Règles du procédé conforme au métamodèle des Règles (à l'image de la *collecte des interfaces* dans l'étape 1 du flot). Dans un deuxième temps, toutes les connexions entre les modèles de simulation des éléments sont créées, c'est l'instanciation des règles. A partir de la Nomenclature_V et du modèle de Règles du procédé sont instanciées les connexions liées à l'instrumentation, tandis que celles liées à la circulation du fluide sont instanciées à partir du modèle des Equipotentiels. Le programme **InstanciationRegle.java** permet ainsi d'obtenir le modèle des règles instanciées du système (ReglesSysInst) conformement au métamodèle d'un Modèle Modelica.

Les connexions liées à la circulation du fluide dans le procédé se font en connectant les interfaces des modèles des éléments dont `kind` vaut *PhyFlow*, *InFlow* ou *OutFlow*. A partir du modèle des Equipotentiels, le programme **InstanciationRegle.java** extrait la liste des équipotentiels, et ne récupère que les extrémités des connexions qui correspondent à un élément. Ainsi, seulement les extrémités ayant un attribut `portNum` sont récupérées. Ces informations permettent par la suite de créer les connexions entre les interfaces correspondantes dans le modèle de simulation. Toutefois, lorsqu'un équipotentiel représente une connexion de plus de deux éléments, la connexion entre les éléments ne peut pas se faire directement. Nous avons donc défini un modèle en langage Modelica permettant de faire la jonction, que nous avons appelé Equipotentiel (Figure 7.21). Ce modèle permet de connecter N éléments entre eux, la valeur de N étant définie lors de l'instanciation de ce modèle.

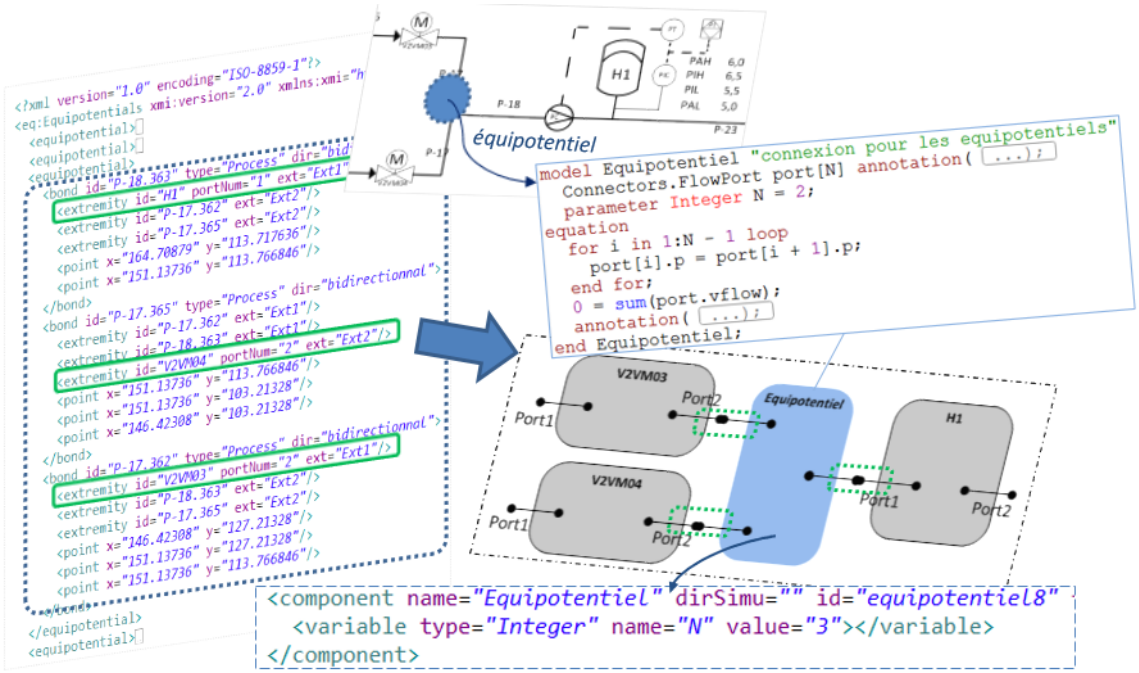


FIGURE 7.21 – Instanciation du modèle d'un équipotentiel

Prenons par exemple l'illustré sur la Figure 7.21. Les extrémités des connexions de l'équipotentiel ayant un attribut `portNum` correspondent au port 1 de H1, au port 2 de V2VM04 et au port 2 de V2VM03. Il s'agit ainsi d'établir une connexion entre V2VM03, V2VM04 et H1, le modèle *Equipotentiel* doit donc être instancié, afin de pouvoir connecter les interfaces des instances V2VM03, V2VM04 et H1. Pour instancier ce modèle, une balise `<component>` est utilisée. L'attribut `name` prend la valeur "Equipotentiel", l'attribut `dirSimu` vaut "" ce modèle n'ayant pas de vue de simulation. La valeur de l'attribut `id` correspondant au nom de l'instance est donnée par ordre de création. Il s'agit ici de la huitième instance d'un équipotentiel. Afin de spécifier la valeur de N de cette instance, une balise `<variable>` est insérée entre `<component> </component>` dont l'attribut `value` vaut 3.

Une fois l'équipotentiel instancié, les connexions entre les interfaces peuvent être créées. Ces connexions se font à l'aide des balises `<connect> </connect>` qui permettent de connec-

ter deux variables d'un élément. Chacune de ces variables est définie par une balise `<element>` dont l'attribut `cptName` correspond au nom de l'instance de l'élément, et `portNum` au numéro du port de l'instance. En reprenant l'exemple de l'équipotentiel entre V2VM03, V2VM04 et H1 (Figure 7.22), le lien entre les instances se fait en connectant les ports de ces instances à ceux de equipotentiel8. Ainsi, le port1 de H1 est connecté au port1 de equipotentiel8, le port2 de V2VM04 est connecté au port2 de equipotentiel8 et le port2 de V2VM03 est connecté au port3 de equipotentiel8 (Figure 7.22).

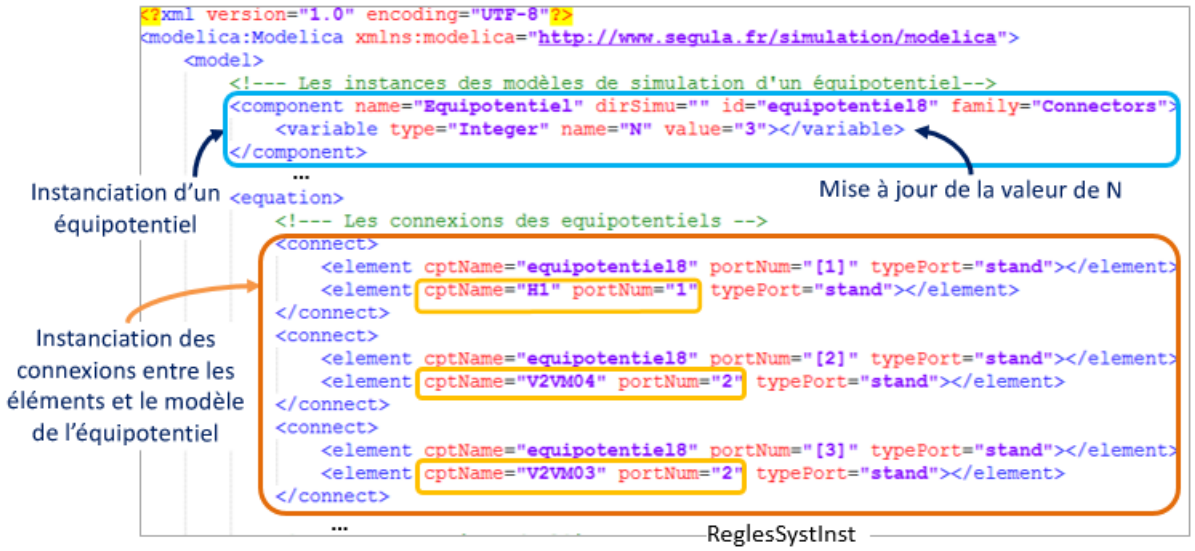


FIGURE 7.22 – Extrait du modèle des règles instanciés du système ReglesSystInst.xml

De façon générale, la connexion entre deux interfaces de simulation, définie par un connecteur (connector) dans le modèle en langage Modelica, se fait via l'utilisation des balises `<connect> </connect>` qui contiennent les deux balises `<element>`. Ainsi, les connexions définies par un équipotentiel, nécessitant ou non l'utilisation d'une instance d'un équipotentiel dans le modèle de simulation, se fait via l'utilisation de balises `<connect>`. Il en va de même pour la connexion des éléments d'instrumentation aux éléments du procédé (Figure 7.24).

Les connexions liées aux éléments d'instrumentation se font en connectant les interfaces des modèles des éléments dont `kind` vaut *MesPhy*. Pour pouvoir identifier les connexions correspondantes dans la Nomenclature_V, le modèle des règles est instancié. Une règle spécifie pour un élément d'instrumentation donné, à quel élément du procédé il peut être connecté. Il est donc nécessaire de déterminer en premier lieu quelles sont les instances de ces éléments, afin d'obtenir les instances des règles.

Le programme **InstanciationRegle.java** identifie ainsi les instances de la Nomenclature_V correspondant à l'élément d'instrumentation de la règle et contenant la balise `<ci>` dont la valeur de l'attribut `nature` correspond à celle de la règle (Figure 7.23). En récupérant l'identifiant de l'instance et la valeur de l'attribut `value` de la balise `<ci>`, les noms du couple d'instance des éléments qui sont associées à une instance de la règle sont connus. Puis, une fois toutes les instances des règles obtenues, le programme identifie les connexions (`<bond>`) de la Nomenclature_V dont les identifiants des extrémités correspondent aux noms des instances

(Figure 7.23). Les connexions des interfaces de simulation correspondantes peuvent ainsi être créées dans le modèle (Figure 7.24).

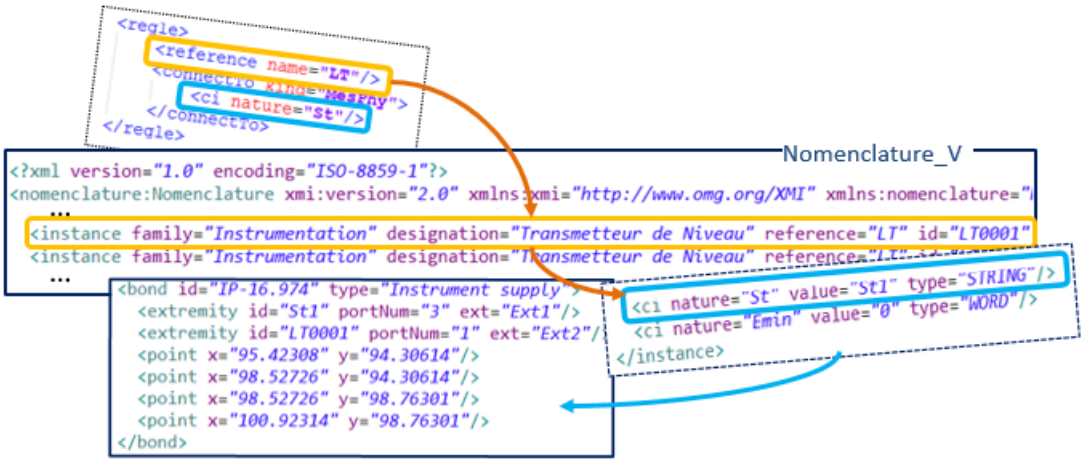


FIGURE 7.23 – Instanciation du modèle de règles

Prenons l'exemple de la Figure 7.23. Pour trouver le nom d'un couple d'instances associées à cette règle, la liste des instances de la Nomenclature_V est parcourue afin de trouver celles dont `reference`="LT". L'instance LT0001 de LT contient une balise `<ci>` dont l'attribut `nature` correspond à celui de la règle (`nature`="St"). Ainsi, une instance de cette règle correspond à la connexion de l'instance LT0001 avec l'instance St1. En parcourant la liste des connexions de la Nomenclature_V, on trouve la connexion associée à l'instance de la règle : la balise `<bond>` contenant une balise `<extremity>` où `id`="St1" et une deuxième où `id`="LT0001". Ainsi, comme illustré sur la Figure 7.24, il est créé une connexion entre l'interface port3 de St1 et l'interface port1 de LT0001 dans le modèle ReglesSysInst.

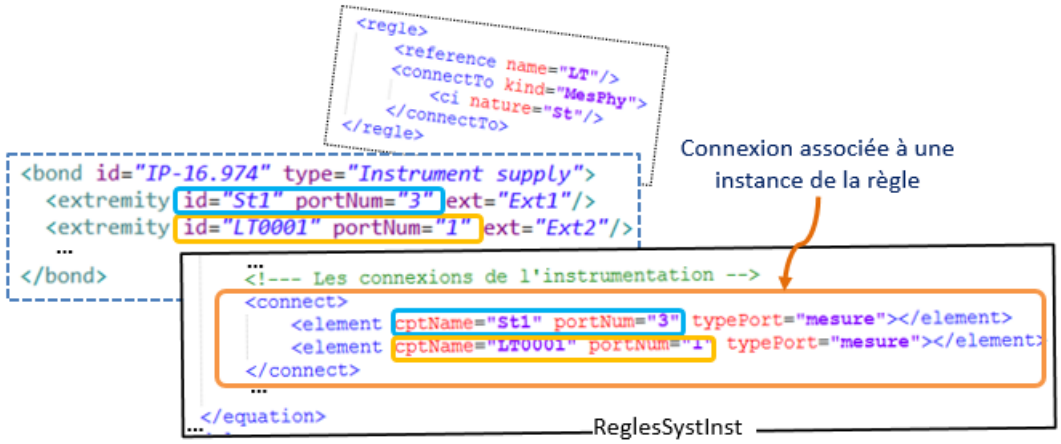


FIGURE 7.24 – Instanciation des connexions de l'instrumentation

Disposant des deux modèles ElementSystInst et ReglesSystInst, il s'agit à présent d'obtenir le modèle de simulation du procédé (sans la communication), objet de l'étape 3c.

7.4.3 Etape 3c. Le modèle de simulation

L'étape 3c est formalisée sur la Figure 7.25. A partir des deux modèles générés lors des étapes 3a et 3b, l'exécution du programme *GenmodelSimu.java* permet de générer un modèle de simulation ModeleSimuSyst conformément au métamodèle d'un Modèle Modelica. Le programme **GenModelSimu.java** génère le modèle ModeleSimuSyst, en insérant le contenu du modèle ReglesSystInst dans le modèle ElementSystInst.

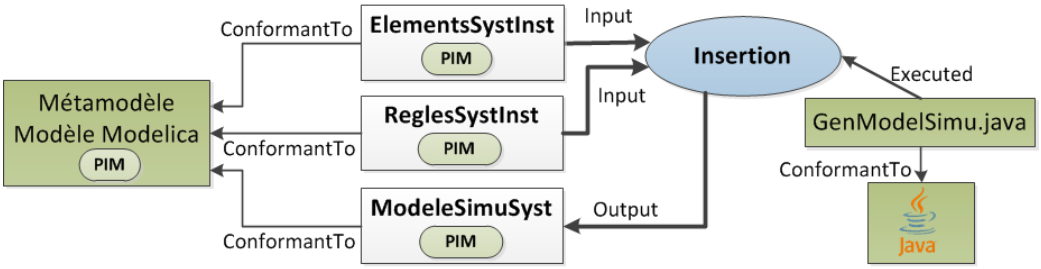


FIGURE 7.25 – Formalisation de l'étape 3c

Un aperçu du squelette du modèle ModeleSimuSyst est donné en Annexe A. La structure du modèle reprend celle du modèle ElementSystInst auxquels ont été rajoutées les informations liées aux connexions (issues du modèle ReglesSystInst). La balise `<model>` contient ainsi toutes les balises `<component>` qu'elles soient liées à l'instanciation des modèles de simulation des éléments ou de celui des équipotentiels. A l'intérieur des balises `<equation> </equation>`, sont contenues toutes les connexions que ce soit concernant les variables globales, les équipotentiels ou l'instrumentation.

7.5 Etape4 : Instanciation de la communication OPC

Rajouter au modèle de simulation la communication avec la partie commande est le but de cette quatrième étape du flot (Figure 7.26). Il s'agit d'instancier la communication à l'aide d'une librairie dédiée à l'utilisation d'un serveur OPC [3]. Puis, de compléter le modèle de simulation afin d'obtenir le modèle de simulation avec la communication OPC.

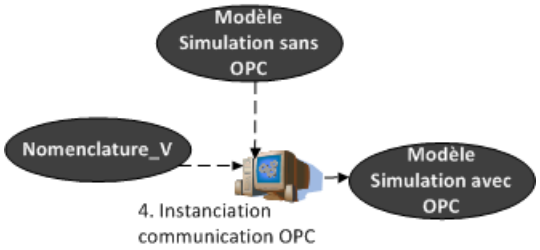


FIGURE 7.26 – Etape4

L'étape 4 est formalisée Figure 7.27. A partir de la Nomenclature_V, le programme *InstanciationOPC.java* génère le modèle VarOPCInst (conformément au métamodèle d'un Modèle Modelica). Puis, le programme *InsertionOPC.java*, à partir des modèles VarOPCInst

3. Dans notre cas, il s'agit de la librairie *OPCClassic* du logiciel SystemModeler.

et ModeleSimuSyst, génère le modèle ModeleSimuComplet (également conforme au métamodèle d'un Modele Modelica).

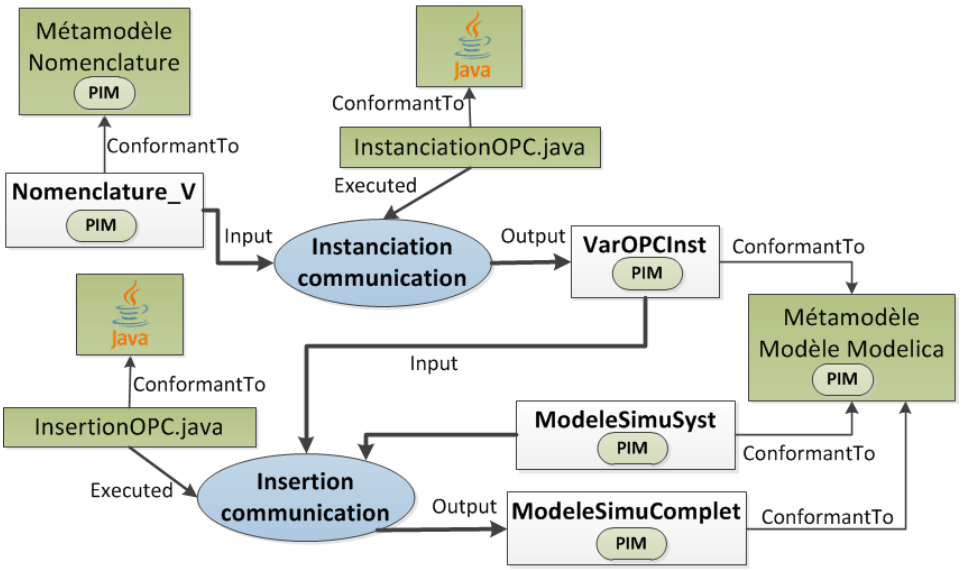


FIGURE 7.27 – Formalisation de l'étape 4

Une fois la structure du modèle VarOPCInst créée, le programme **InstanciationOPC.java** instancie le modèle du serveur OPC. Cette instance du serveur OPC correspond à la première balise `<component>` (Figure 7.28). La valeur de l'attribut `name` indique qu'il s'agit d'une instance du modèle *OPCServer* de la librairie OPC.


```
<?xml version="1.0" encoding="UTF-8"?>
<modelica:Modelica xmlns:modelica="http://www.segula.fr/simulation/modelica">
  <model>
    <component name="OPCServer" dirSimu="" id="opcServer" family="OPCS">
    <component name="SamplerReal.Write" dirSimu="" id="Mes_LS0001" family="OPC">
    <component name="SamplerReal.Write" dirSimu="" id="Mes_LS0002" family="OPC">
    <component name="SamplerInteger.Write" dirSimu="" id="Mes_LT0001" family="OPC">
    <component name="SamplerInteger.Write" dirSimu="" id="Mes_LT0002" family="OPC">
      <variable type="String" name="itemName" value="Channel1.Device1.Mes_LT0002"></va
    </component>
    <component name="SamplerBoolean.Read" dirSimu="" id="CmdO_V2VM01" family="OPC">
      <variable type="String" name="itemName" value="Channel1.Device1.CmdO_V2VM01"></v
    </component>
    <component name="SamplerBoolean.Read" dirSimu="" id="CmdC_V2VM01" family="OPC">
    <component name="SamplerBoolean.Write" dirSimu="" id="FdcO_V2VM01" family="OPC">
```

FIGURE 7.28 – Extrait du modèle VarOPCInst : instanciation des variables OPC

Débute alors l'instanciation des variables OPC. Pour ce faire, les interfaces de simulation de la Nomenclature_V dont `kind`="InPhy" ou `kind`="OutPhy" sont récupérées. La sélection du modèle à instancier étant dépendante du type de la variable (réel, entier, booléen...), et de son mode d'accès au serveur (lecture ou écriture), elle se fait à partir des attributs `type` et `kind` d'une interface de simulation. En effet, le mode lecture d'une variable OPC correspond à une interface dont `kind`="InPhy", et le mode écriture à `kind`="OutPhy". De plus, la structure de la librairie OPC est telle, que pour chaque type de variable OPC correspond un *package* contenant un modèle *Read* pour le mode lecture et un modèle *Write* pour le mode écriture. Ainsi, à partir de l'attribut `type` de l'interface est sélectionné le *package* dans lequel se situe

le modèle à instancier.

Suite à l'instanciation des variables OPC, les connexions entre ces variables et les interfaces de simulation des instances des éléments doivent être créées. De même, les variables OPC doivent être connectées à l'instance du serveur OPC. Ces connexions sont créées dans le modèle via les balises `<connect>` à l'intérieur de la balise `<equation>` (Annexe B). Une fois ces connexions créées, la phase d'instanciation de la communication est terminée, nous disposons alors du modèle VarOPCInst. La phase d'insertion de la communication peut alors débuter. A partir du modèle VarOPCInst, le programme **InsertionOPC.java** complète le modèle ModeleSimuSyst, générant ainsi le modèle de simulation complet.

Un aperçu du squelette du modèle ModeleSimuComplet est donné Figure 7.29. La structure reprend celle du modèle ModèleSimuSyst auquel sont ajoutés les informations issues du modèle VarOPCInst. Ainsi, la balise `<model>` contient toutes les balises `<component>` qu'elles soient liées à l'instanciation de la communication OPC, des modèles de simulation des éléments du procédés ou des équipotentiels. A l'intérieur des balises `<equation></equation>` sont contenues toutes les connexions que ce soit par rapport aux variables globales, à la circulation du fluide, à l'instrumentation ou bien par rapport aux échanges avec la partie commande.

```
<?xml version="1.0" encoding="UTF-8"?>
<modelica:Modelica xmlns:modelica="http://www.segula.fr/simulation/modelica">
  <package domain="PlateForme">
    <package system="Auxiliaires">
      <model subSystem="EdS">
        <!-- Les instances des modèles pour la communication OPC -->
        <component name="OPCServer" dirSimu="" id="opc
          <variable type="String" name="serverName" va
        </component>
        <component name="Sa
          <variable type="St
        </component>
        ...
        <equation>
          <!-- Les connexions entre variables OPC et les interfaces -->
          <connect>
              <element cptName="Mes_LT0002.u" typePort="OPCV"></element>
              <element cptName="LT0002.Mes" typePort="OPCV"></element>
          </connect>
          <!-- Les connexions entre variables OPC et le serveur OPC -->
          <connect>
              <element cptName="Mes_LT0002" typePort="SRV"></element>
              <element cptName="opcServer" typePort="SRV"></element>
          </connect>
        </equation>
      </model>
    </package>
  </package>
</modelica:Modelica>
```

FIGURE 7.29 – Extrait du squelette du modèle de simulation ModeleSimuComplet.xml

Nous disposons à présent du modèle de simulation complet du procédé, indépendant de toute plateforme d'exécution.

7.6 Etape5 : Génération du code Modelica

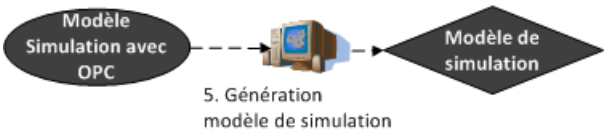


FIGURE 7.30 – Etape5

Cette cinquième et dernière étape du flot (Figure 7.30) vise à obtenir le modèle de simulation en langage Modelica, qui pourra être compilé puis exécuté. Il s'agit ainsi de transformer le modèle de simulation au format xml en code Modelica. **L'étape 5** est formalisée sur la Figure 7.31. A partir du modèle ModeleSimuComplet conforme au métamodèle d'un Modele Modelica, le programme *GeneCodeModelica.java* génère le modèle ModeleSimu en langage Modelica.

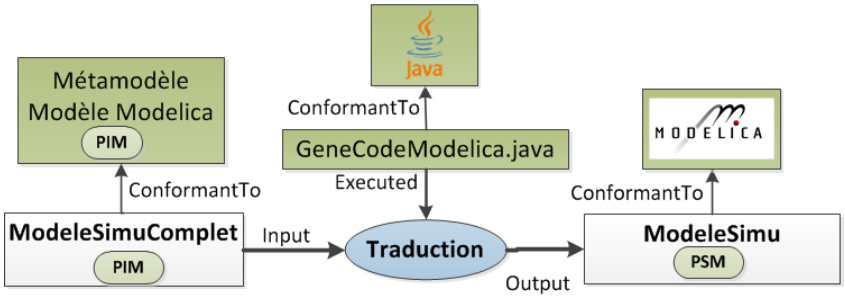


FIGURE 7.31 – Formalisation de l'étape 5

Le programme **GeneCodeModelica.java** commence par créer le modèle ModeleSimu (écriture de la première ligne du code). Cette première ligne du code débute par le mot-clé `model` suivi du nom du modèle. Le nom du modèle est défini à partir de l'attribut `subSystem` de la balise `<model>` du modèle ModeleSimuComplet (Figure 7.32). Puis, le ModeleSimuComplet est parsé afin de rajouter le code associé au modèle de simulation (déclaration des variables globales, instanciation des modèles, connexions...) .

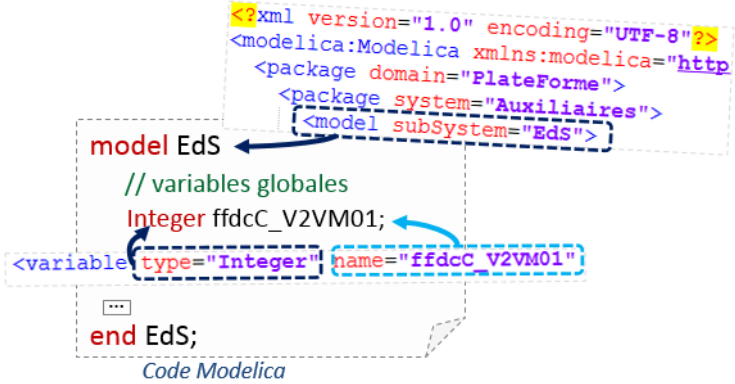


FIGURE 7.32 – Traduction en code Modelica : les variables globales

La déclaration d'une variable, dans le langage Modelica, se fait en donnant son type (par exemple : Integer pour un entier, Boolean pour un booléen...) suivi du nom de la variable.

Remarque 1 (Déclaration d'une variable)
`Type` *nomVariable* `;`
où `Type` est le type de la variable, et *nomVariable* est le nom donné à la variable.

Ces informations sont obtenues à partir des attributs des balises `<variable>` de `<model>`. La valeur de `Type` correspond à celle de l'attribut `type`, et *nomVariable* à celle de `name`. Par exemple, prenons la balise `<variable>` où `type`="Integer" et `name`="ffdcC_V2VM01" (représentée sur la Figure 7.32). La ligne de code associée à la déclaration de cette variable est ainsi « `Integer ffdcC_V2VM01;` ».

Ensuite, le code lié aux instances de modèles est ajouté. La déclaration d'une instance, dans le langage Modelica, se fait, dans le cas général, en donnant la localisation du modèle à instancier suivi du nom de l'instance.

Remarque 2 (Instanciation d'un modèle Modelica)
`LocalisationModele` *nomInstance* `;`
où `LocalisationModele` est la localisation du modèle à instancier, et *nomInstance* est le nom donné à l'instance.

Ainsi, les instances du modèle d'un élément du procédé sont définies en donnant la localisation du modèle dans la librairie de simulation générée, information qui est récupérée à partir des attributs `family` et `name` des balises `<component>`. Par exemple, dans le cas de l'instance V2VM01 du modèle V2VM (Figure 7.33), `LocalisationModele` correspond à la localisation de V2VM dans la librairie Lib_Modelica (à savoir, dans le `package` Process du `package` Elements). En conséquence, la ligne de code associée à l'instance V2VM01 est « `Lib_Modelica.Elements.Process.V2VM V2VM01;` ».

FIGURE 7.33 – Traduction en code Modelica : les instances des éléments

Toutefois, il peut être nécessaire de définir (ou redéfinir) la valeur de variables du modèle à instancier. La valeur de la variable est donnée dans ce cas en argument de l'instance.

Remarque 3 (Instanciation d'un modèle Modelica avec valeur de variables)
`LocalisationModele` *nomInstance* (*nomVariable*=*valeur*) `;`
où *nomVariable* est le nom de la variable, et *valeur* est la valeur donné à la variable.

Ceci correspond à une balise `<component>` (dans ModeleSimuComplet) contenant une balise `<variable>`. Les attributs de la balise `<variable>` spécifient en particulier le nom de la variable (attribut `name`) et la valeur à donner en argument (attribut `value`). Ainsi, *nomVariable* correspond à la valeur l'attribut `name` de `<variable>` et *valeur* à celle de l'attribut `value`.

Tel est le cas, par exemple, d'une instance du modèle d'un équipotentiel où la valeur de la variable N (précisant le nombre d'éléments de l'équipotentiel) doit être mise à jour. L'instance d'un équipotentiel (balise `<component>` dont `name`="Equipotentiel") contient ainsi une balise `<variable>` dont `name`="N" et l'attribut `value` précise la valeur à donner à N. Par exemple, l'instance equipotentiel8 (Figure 7.34) représente une connexion entre trois éléments (`value`="3"). Le modèle étant situé dans le `package` Connectors du `package` Utilitaire de Lib_Modelica, la ligne de code associée à l'instance equipotentiel8 est par conséquent « `Lib_Modelica.Utilitaire.Connectors.Equipotentiel equipotentiel8(N=3);` ».

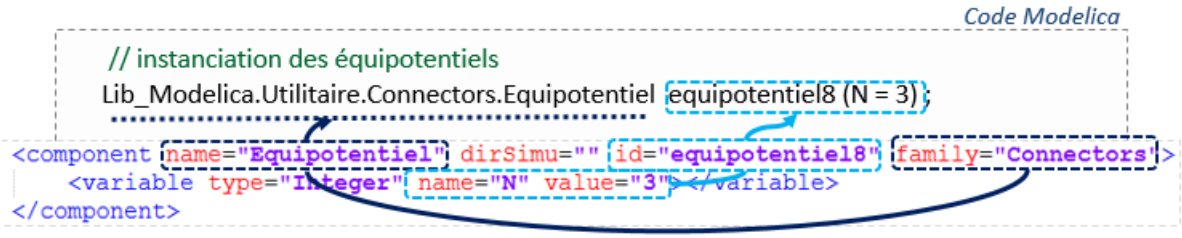


FIGURE 7.34 – Traduction en code Modelica : instance d'un équipotentiel

Une fois que le code associé à toutes les balises `<component>` de ModeleSimuComplet rajouté dans le modèle de simulation, il s'agit d'y inclure celui associé aux connexions (contenu à l'intérieur les balises `<equation> </equation>`. Ce code est rajouté dans la section `equation` du modèle de simulation. Comme évoqué lors du chapitre 6, il existe deux types d'équation : l'équation de forme *classique* et celle de type *connect*.

Le premier cas correspond à une connexion définie à l'aide d'une balise `<calcul>`. Le code associé est récupéré de l'attribut `value`. Il s'agit typiquement de connexion d'une variable globale à l'interface d'une instance d'un modèle d'un élément. Prenons, par exemple, la balise `<calcul>` dont l'attribut `value` vaut "ffdcC_V2VM01 = V2VM01.ffdcC". Le code associé est ainsi « `ffdcC_V2VM01 = V2VM01.ffdcC;` », comme illustré sur la Figure 7.35.

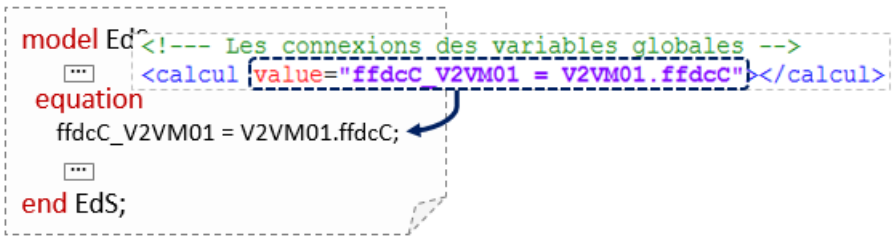


FIGURE 7.35 – Traduction en code Modelica : les connexions des variables globales

Pour une connexion définie à l'aide d'une balise `<connect>`, le code correspondant débute par le mot-clé `connect` auquel sont données en argument les variables à connecter.

Remarque 4 (Equation *connect*)
connect(`variable1 ,variable2` **);**
où `variable1` est la variable à connecter à la `variable2`.

Considérons une connexion d'un flux physique (représentant la circulation du fluide), telle une

connexion liée à un équipotentiel. En l'occurrence, il s'agit d'une connexion d'un port d'une instance de l'équipotentiel à un port d'une instance d'un élément. Par exemple Figure 7.36, le port[1] de l'instance equipotentiel8 doit être connecté au port1 de l'instance H1. Dans ce cas, `variable1` correspond à `equipotentiel8.port[1]` et `variable2` à `H1.port1`. Le code associée à cette connexion est ainsi « `connect(equipotentiel8.port[1],H1.port1);` ».

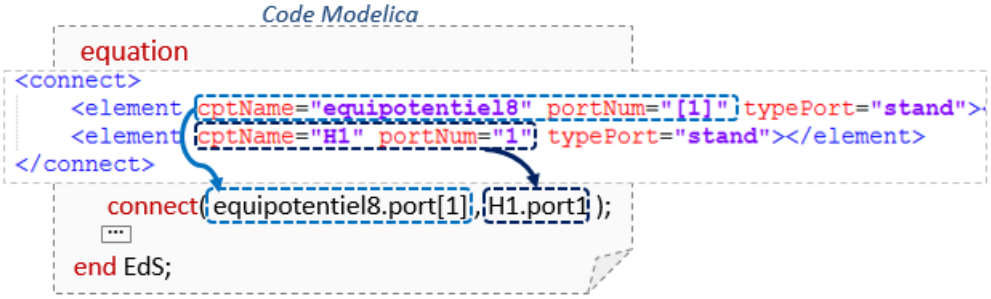


FIGURE 7.36 – Traduction en code Modelica : les connexions *connect*

Une fois que le modèle ModeleSimuComplet a été parsé, le programme rajoute une dernière ligne de code afin d'achever l'écriture du modèle en langage Modelica. Il s'agit du mot-clé `end` suivi du nom du modèle et du symbole `";"` (cf Figure 7.32). Ainsi s'achève la génération du modèle de simulation.

7.7 Validation du *flot de simulation*

Maintenant que l'implémentation du *flot de simulation* a été présentée, nous allons nous intéresser à son utilisation sur un cas d'application.

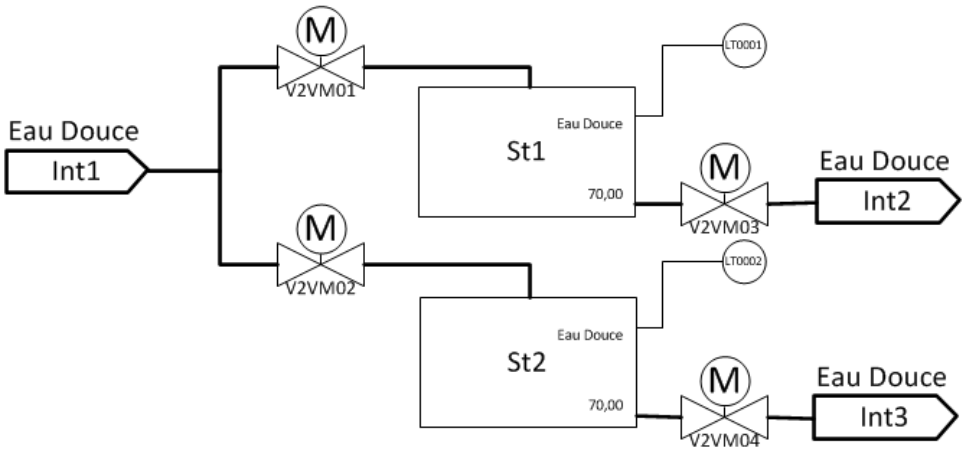


FIGURE 7.37 – Schéma P&ID du cas d'application

Pour ce faire nous considèrerons le procédé représenté sur la Figure 7.37. Ce procédé est composé de quatre vannes à deux voies motorisées (pilotées par le système de conduite), de deux soutes, et d'un transmetteur de niveau pour chaque soute. L'interface *Int1* représentant une arrivée d'eau douce, selon la position des vannes *V2VM01* et *V2VM02*, les soutes St1 et St2 peuvent être remplies. La vidange des soutes dépend quant à elle des positions des vannes

V2VM03 et *V2VM04*. Le niveau dans les soutes est communiqué à la partie commande par les transmetteurs de niveau (*LT0001* et *LT0002*) qui effectuent les mesures.

Il s'agit, dans un premier temps, de générer le modèle de simulation de ce procédé, puis, de le simuler dans un deuxième temps, et ce afin d'en valider le bon fonctionnement.

7.7.1 Application du *flot de simulation*

Afin d'appliquer le flot de simulation, il nous faut disposer du modèle Synoptique et du modèle des Equipotentiels du procédé.

FIGURE 7.38 – Extrait du fichier Synoptique.xml et Equipotentiel.xml

Ainsi, à partir du schéma P&ID saisi sous Microsoft Visio, nous obtenons le modèle Synoptique (à gauche, Figure 7.38) suite à l'exécution de l'opération d'épuration du flot de conception d'Anaxagore [Bignon, 2012]. De même, à partir du flot de conception d'Anaxagore, il est généré le modèle des Equipotentiels (à droite, Figure 7.38). Le flot de simulation peut maintenant être appliqué.

En appliquant le flot de simulation, la librairie de simulation et le modèle de simulation du procédé sont ainsi générés (Figure 7.39). Ces derniers sont générés en seulement quelques secondes, tandis qu'une vingtaine de minutes est nécessaire pour construire manuellement ce modèle de simulation, à partir de la librairie générée. Il est à noter que le modèle contient déjà plus de 200 équations.

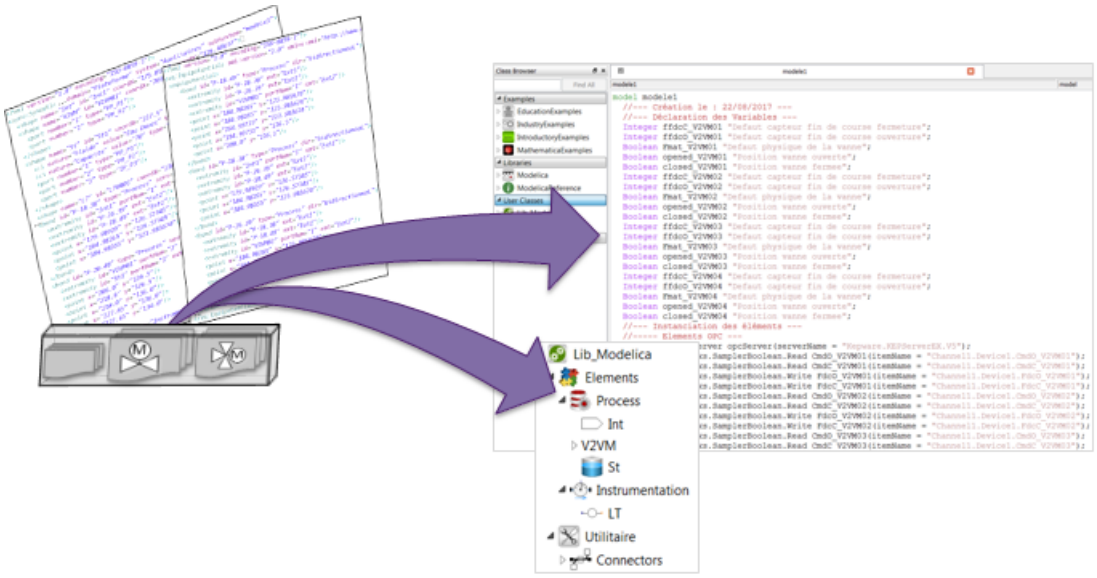


FIGURE 7.39 – Génération de la librairie de simulation et du modèle de simulation du procédé

7.7.2 Simulation du procédé

Afin de tester le bon fonctionnement du modèle généré, nous nous proposons dans un premier temps de remplir la soute St1, puis de la vider. Pour ce faire, la vanne V2VM01 doit être ouverte pour que la soute se remplisse tandis que la vanne V2VM03 doit être fermée. Puis, afin de vider la soute, il s'agit d'ouvrir la vanne V2VM03 tandis que la vanne V2VM01 doit être fermée. Après avoir initialisé le modèle de simulation (vannes en positions fermées et soutes vides), la simulation peut être lancée.

Dans un premier temps, intéressons-nous aux échanges entre partie commande et simulation, en se plaçant au niveau du programme de commande. Un aperçu du programme de commande est donné Figure 7.40, après le démarrage de la simulation. Toutes les vannes sont ainsi fermées comme indiqué par les variables associées aux interfaces FdcC des instances des vannes (représentant la valeur du capteur de fin de course de fermeture). En effet, la couleur rouge des variables (Figure 7.40) correspond à la valeur `true`. La valeur des interfaces FdcO (représentant la valeur du capteur de fin de course d'ouverture) ont quant à elles la valeur `false` (représentée en bleu). D'autre part, on peut noter qu'à cet instant il n'y a ni demande d'ouverture de vannes (à partir de l'interface de supervision), ni de commande d'ouverture (envoyée au procédé). Les variables CtrlO, CmdO et CmdC de chaque vanne ont ainsi la valeur `false`.

Une demande d'ouverture de la vanne V2VM01 est effectuée à partir de l'interface de supervision. Le programme de commande recevant cette requête (passage de la valeur de la variable *CtrlO_V2VM01* de `false` à `true`), envoie alors une commande d'ouverture au procédé simulé (CmdO_V2VM01 vaut alors `true`). Après réception de la commande d'ouverture de V2VM01, la vanne V2VM01 s'ouvre (FdcC_V2VM01 passe à `false`, et une fois la position ouverte atteinte, la variable FdcO_V2VM01 prend la valeur `true`). La Figure 7.41 montre l'état du programme de commande après que la vanne ait atteint sa position ouverte. Ce der-

nier a bien reçu les changements de valeurs envoyés par le procédé simulé (FdcO_V2VM01 apparait maintenant en rouge, tandis que FdcC_V2VM01 apparait en bleu).

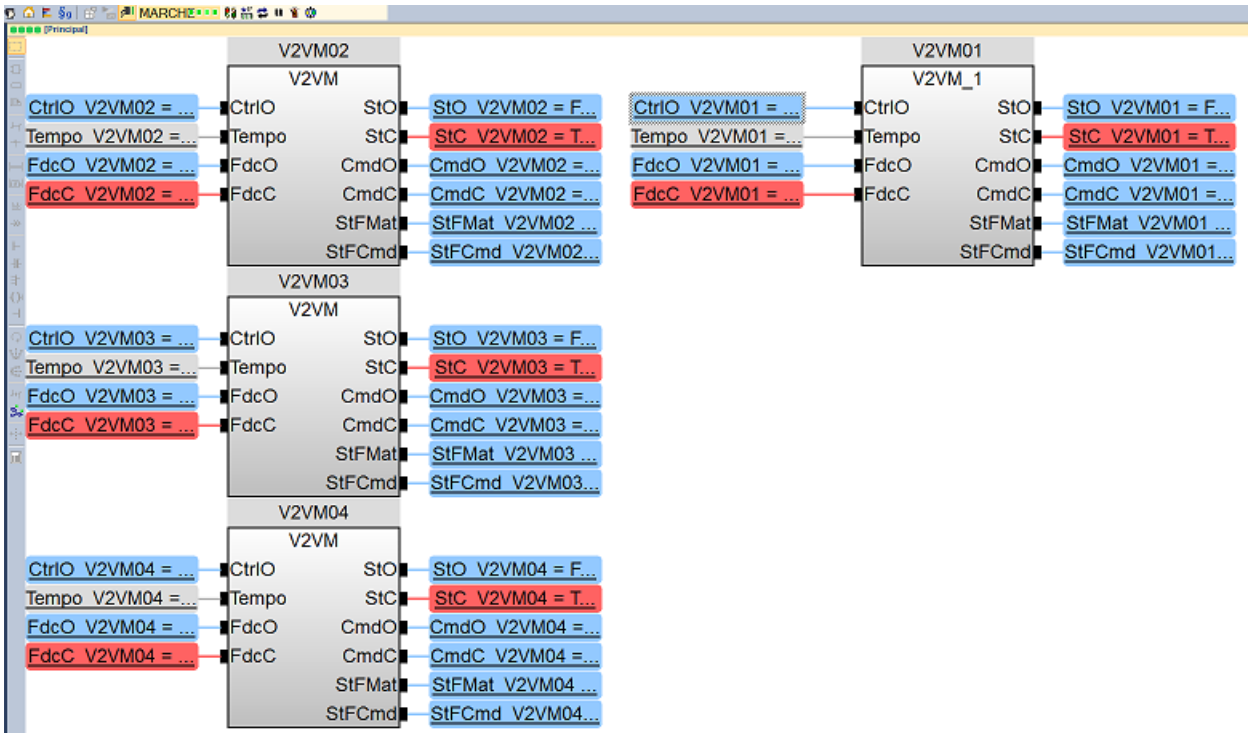


FIGURE 7.40 – Programme de commande au démarrage de la simulation

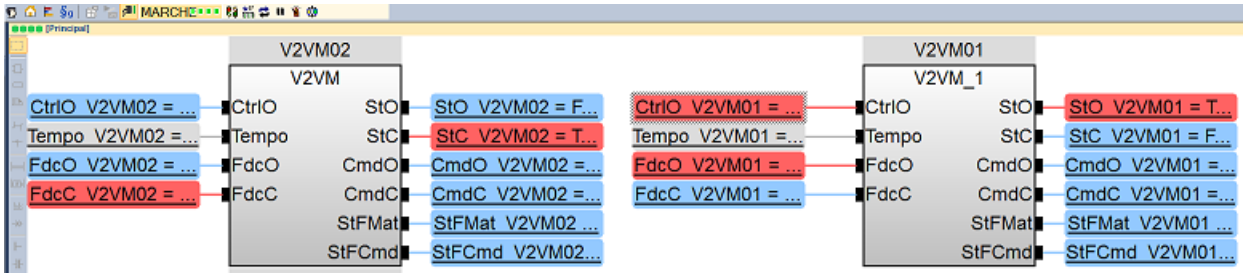


FIGURE 7.41 – Programme de commande suite à une demande d'ouverture de V2VM01

Puis, une fois que la soute est suffisamment remplie, la fermeture de la vanne V2VM01 est demandée via l'interface de supervision (CtrlO_V2VM01 repasse alors à `false`), ce qui déclenche l'envoi d'une commande de fermeture (CmdO_V2VM01 a alors la valeur `true`). La vanne entame sa phase de fermeture, FdcO_V2VM01 passe alors à `false`. Et une fois la vanne fermée, la valeur de FdcC_V2VM01 repasse à `true`. Ensuite, afin de vider la soute, une demande d'ouverture de la vanne V2VM03 est envoyée au programme de commande, qui envoie alors la commande d'ouverture au procédé simulé. Suite à celle-ci, la vanne quitte sa position fermée (FdcC_V2VM03 passe alors à `false`), et une fois sa position ouverte atteinte, la variable FdcO_V2VM03 passe à `true`. On se retrouve alors dans la configuration représentée sur la Figure 7.42 (toutes les vannes sont en position fermée, à l'exception de la

vanne V2VM03 qui est en position ouverte).

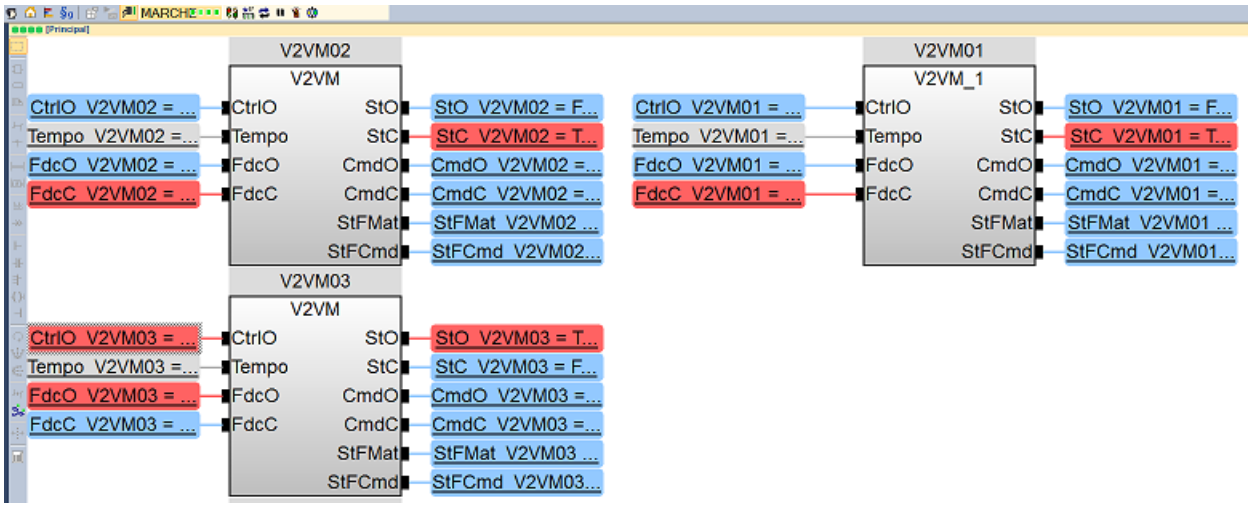


FIGURE 7.42 – Programme de commande suite à la demande d'ouverture de V2VM03

Ainsi, du point de vue de la partie commande, le modèle de simulation semble jouer son rôle, en communiquant à la partie commande les valeurs des positions de fin de course des vannes en fonction des commandes qui sont reçues. Il s'agit maintenant de s'assurer que l'évolution du comportement du procédé est correctement modélisée.

Les résultats de la simulation obtenus sont représentés sur la Figure 7.43. La courbe en bleu correspond à l'évolution du niveau dans la soute. La valeur associée à la position ouverte de la vanne V2VM01 est représentée en orange, tandis que celle associée à la vanne V2VM03 est représentée en vert.

Nous constatons que, suite à l'ouverture de la vanne V2VM01, le niveau dans la soute augmente et qu'il cesse d'augmenter suite à la fermeture de la vanne. Il est également à noter que le niveau dans la soute commence à croître un peu avant que la position ouverte soit atteinte. Ceci est dû au fait qu'il est considéré, dans la modélisation de la vanne de Niveau Contextualisation, que dès lors que la vanne n'est plus en position fermée la vanne laisse passer le fluide. Ce décalage correspond ainsi au temps de manoeuvre de la vanne. De même, tant que la vanne n'a pas atteint sa position fermée, la soute continue de se remplir.

Ensuite, la vanne V2VM03 quitte sa position fermée, à cet instant le niveau dans la soute commence à diminuer. C'est ainsi que débute la phase de vidange de la soute, celle-ci se terminant avec la fermeture de la vanne. Ainsi, lorsque que la position fermée de la vanne V2VM03 est atteinte, le niveau dans la soute devient constant. Le comportement observé du procédé simulé est ainsi cohérent avec le scénario de test. Nous pouvons ainsi conclure que le modèle de simulation généré se comporte comme attendu.

Au travers de cet exemple, nous avons montré que l'implémentation de notre proposition de *flot de simulation* permettait d'obtenir un modèle de simulation qui fonctionne correctement. Et ceci que ce soit par rapport aux échanges avec la partie commande (communication OPC) ou bien l'état du procédé.

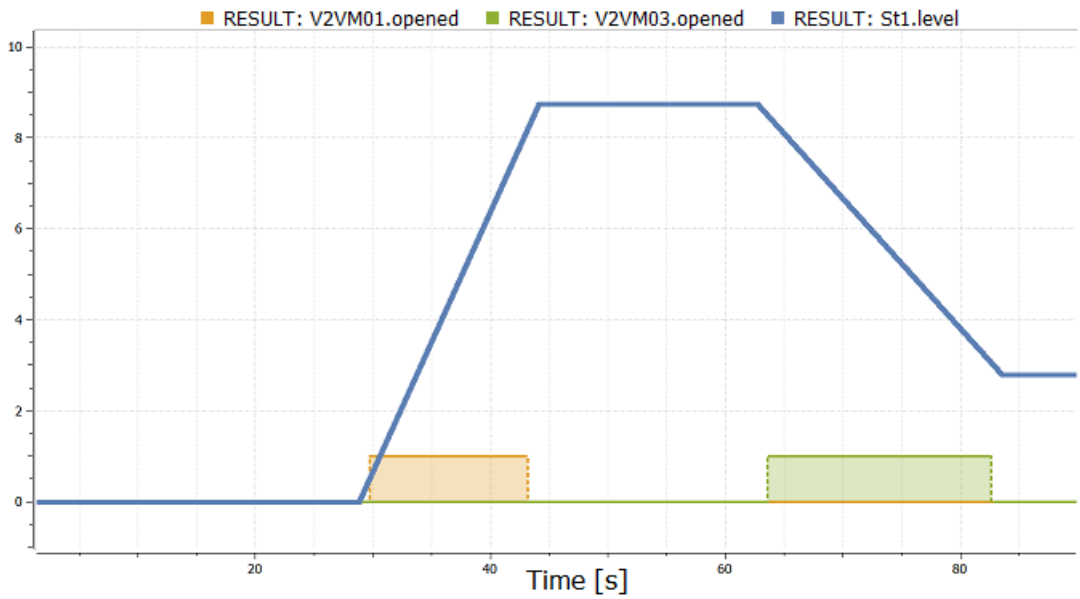


FIGURE 7.43 – Résultat de la simulation

7.8 Bilan de l'implémentation du flot

Afin d'apporter une preuve de concept, nous avons présenté l'implémentation de notre proposition de *flot de simulation*. Puis, nous l'avons appliquée sur un exemple comportant quatre types d'éléments différents (soute, vanne à deux voies motorisée, transmetteur de niveau et interface de procédé). A partir du schéma P&ID (comportant 10 instances d'éléments), la librairie de simulation et le modèle de simulation associés ont été générés **en moins de dix secondes** (au lieu d'une vingtaine de minutes manuellement). Ce modèle de simulation comportait 140 lignes de code Modelica dont 30 instances de modèles, 20 variables d'entrée/sortie internes à la simulation. La section `equation` du modèle contient quant à elle 79 équations dont 47 de type `connect`.

L'application du *flot de simulation* sur cet exemple a ainsi permis de mettre en évidence le gain de temps dans l'obtention du modèle de simulation. Puis, pour apporter une preuve quant à la validité du modèle généré, nous l'avons testé sur un scénario. L'objectif était double : il s'agissait d'une part de s'assurer du bon fonctionnement des échanges avec la partie commande (communication OPC), mais également de s'assurer de l'évolution correcte de l'état du procédé simulé.

Cette mise à l'épreuve fut un succès : non seulement nous sommes capable de simuler le comportement du procédé, mais également de configurer correctement la communication avec le serveur OPC (et donc la partie commande). Ce dernier point est notable, car la configuration de la communication avec le serveur OPC est une tâche assez fastidieuse lorsqu'elle est réalisée manuellement, et souvent source d'erreur. Or, via l'intégration du *flot de simulation* en parallèle du flot de conception d'Anaxagore, nous sommes en mesure de garantir la cohérence syntaxique pour la communication avec la partie commande. Ainsi, l'obtention du modèle de simulation est facilitée par rapport au temps (la génération se fait en quelques

seconde), et par rapport à la communication OPC.

L'implémentation de notre proposition de *flot de simulation* permet ainsi d'apporter une preuve quant à la validité de notre approche de génération automatisée de modèles de simulation. Au travers de notre contribution nous avons facilité l'obtention de modèles de simulation, quelques secondes sont nécessaires pour passer du schéma P&ID à un modèle de simulation en langage Modelica, incluant la communication avec la partie commande.

Toutefois, bien que l'obtention soit facilitée, la mise en oeuvre des tests du système de conduite requiert néanmoins une étape d'initialisation du modèle de simulation manuelle. En effet, le modèle généré est indépendant de l'état initial du procédé spécifié dans un scénario de test, il faut avant de lancer une simulation paramétrer manuellement le modèle de simulation.

Maintenant que nous avons apporté une preuve de concept de notre *flot de simulation*, l'objet du chapitre suivant concernera la mise en oeuvre de la démarche de vérification.

8

Preuve d'usage : Application de la démarche de vérification

Sommaire

Afin de permettre l'intégration de tests dès le début de la conception d'un système de conduite, nous avons proposé une approche de génération automatisée de modèles de simulation du procédé (chapitre 6). Nous avons également formalisé une méthodologie afin de faciliter l'analyse des résultats issus de la simulation (chapitre 4). Via l'implémentation du *flot de simulation* (chapitre 7) nous avons apporté une preuve de concept, il s'agit maintenant d'appliquer la démarche de vérification en vue d'obtenir une preuve d'usage.

Dans ce chapitre, nous nous intéressons à l'application de nos propositions dans le cadre du flot de conception automatisée d'Anaxagore. Un intérêt sera également porté sur la mise en place de vérifications au plus tôt dans ce flot.

Par soucis de clarté et de concision, nous appliquerons la démarche de vérification sur un cas d'étude relativement simple, basé sur celui du chapitre 7. Les résultats sur un système industriel complet seront ensuite présentés plus succinctement, afin de montrer la portée de notre approche et la faisabilité du passage à l'échelle sur des systèmes complexes.

8.1 Application de la démarche de vérification

De façon générale, dans le flot de conception d'Anaxagore, les tests par simulation peuvent s'effectuer à trois niveaux (Figure 8.1).

- Une première série de vérifications peut s'effectuer suite à l'étape de génération de bas niveau du système de contrôle-commande. Il s'agit de vérifier la chaîne de contrôle-commande d'un élément (par exemple : une vanne), et également de vérifier le bon fonctionnement du système d'alarmes (liées à l'instrumentation, par exemple).
- Une deuxième série de vérifications peut être effectuée suite à l'étape de spécification des fonctions de haut niveau. Il s'agit dans ce cas de vérifier que les séquences de commandes élémentaires spécifiées permettent bien d'emmener le procédé dans un état qui délivre le service.
- Enfin, une troisième série de vérifications peut être effectuée suite à la génération de haut niveau. Il s'agit dans ce cas, de tester l'implémentation des fonctions de haut niveau, et le caractère reconfigurable du système.

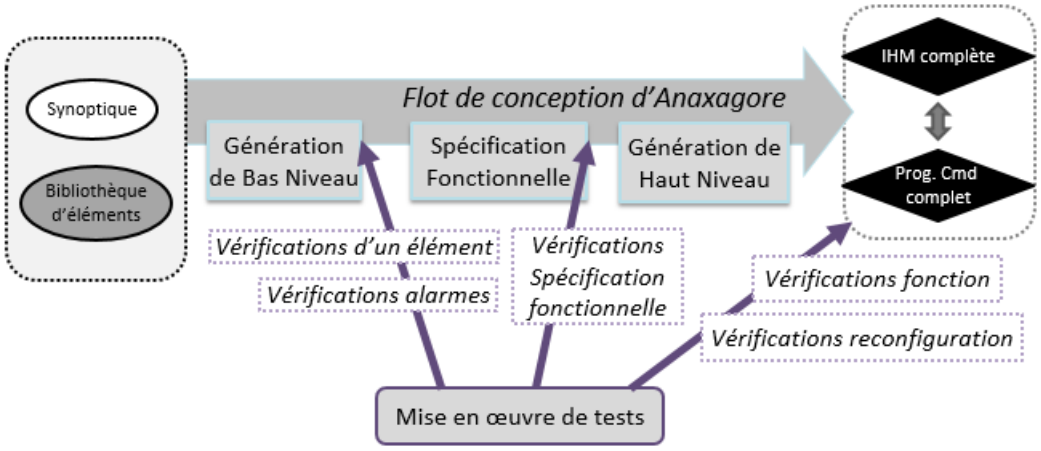


FIGURE 8.1 – La place des vérifications

Dans les paragraphes suivants, nous présentons une première application de notre démarche. Il s'agit notamment de démontrer l'intérêt de nos propositions sur la mise en oeuvre de la première série de tests, qui concerneront plus particulièrement le système d'alarmes liées à l'instrumentation.

8.1.1 Le cas d'étude

Le cas d'étude considéré pour la mise en oeuvre de vérifications est représenté sur la Figure 8.2. Celui-ci est similaire au cas d'étude considéré au chapitre 7 (Figure 7.37), auquel il a été ajouté des informations liées à la supervision et des détecteurs de trop-plein. Ce système est ainsi constitué de deux soutes St1 et St2 qui peuvent contenir 70m^3. Chacune d'elles dispose d'une vanne de remplissage (V2VM01 pour St1, et V2VM02 pour St2) et d'une vanne de vidange (V2VM03 pour St1, et V2VM04 pour St2). Comme dans le cas d'application du chapitre 7, un transmetteur de niveau est associé à chaque soute (LT0001 pour St1, et LT0002 pour St2). Différents seuils d'alarme sont associés à la mesure de niveau dans les soutes : niveau très bas (LAL), niveau bas(LIL), niveau haut(LIH) et niveau très haut(LAH). Chaque soute dispose également d'un détecteur de trop-plein (LS0001 pour St1 et LS0002 pour St2), qui s'intègre dans une boucle de réaction qui, le cas échéant, envoie une commande électrique de fermeture à la vanne de remplissage.

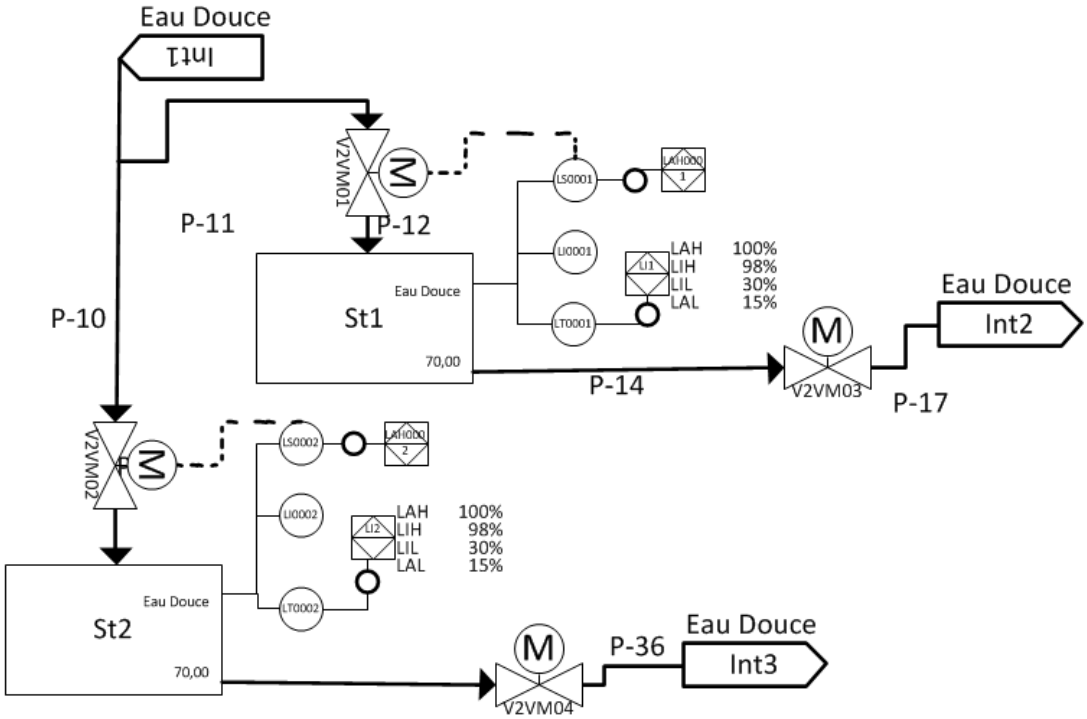


FIGURE 8.2 – Schéma P&ID du cas d'étude

Le cas d'étude ayant été présenté, il s'agit maintenant d'appliquer la démarche de vérification présentée au chapitre 4. L'objet de la section suivante concerne ainsi la modélisation des propriétés.

8.1.1.1 Modélisation des propriétés

Le but d'un tel système est de maintenir la disponibilité en eau potable à bord du navire. Ce système contribue également à la stabilité du navire. Pour y parvenir, le système peut stocker de l'eau, distribuer de l'eau ou remplir une soute. En appliquant le formalisme de modélisation multi-niveau (cf chapitre 4), nous obtenons la modélisation des propriétés de ce système, à différents niveaux de décomposition. La modélisation au niveau de décomposition *Système* est représentée sur la Figure 8.3. Le stockage peut être mis en oeuvre par un Stockage dans la soute St1 et/ou St2. De même, la distribution peut se faire à partir de la soute St1 et/ou St2, et le remplissage peut concerner la soute St1 et/ou St2. Ainsi, pour atteindre les objectifs, il est possible d'effectuer un Stockage (qui peut être implémenté par un Stockage St1 et/ou Stockage St2), un Remplissage (Remplissage St1 et/ou Remplissage St2), ou une Distribution (Distribution St1 et/ou Distribution St2).

Un extrait de la modélisation des propriétés au niveau de décomposition *Fonction* est donné Figure 8.4. Pour effectuer un Stockage St1 (respectivement Stockage St2), une opération de Stockage Passif doit être réalisée par la soute St1 (respectivement St2) ainsi qu'une opération de Stockage Actif par la vanne V2VM03 (respectivement V2VM04). Afin d'effectuer un Remplissage St1 (respectivement Remplissage St2), une fonction de Routage de Remplissage St1 (respectivement Routage de Remplissage St2) et une fonction de Routage de Stockage

St1(respectivement Routage de Stockage St2) doivent être mises en oeuvre. Il est à noter que la fonction de Routage de Remplissage St1 est mise en oeuvre par une opération de Libre Passage réalisée par la vanne V2VM01, et potentiellement par une opération de Stockage Actif réalisée par la vanne V2VM02. La fonction de Routage de Remplissage St2 est quant-à-elle mise en oeuvre par une opération de Libre Passage réalisée par la vanne V2VM02, et potentiellement par une opération de Stockage Actif réalisée par la vanne V2VM01.

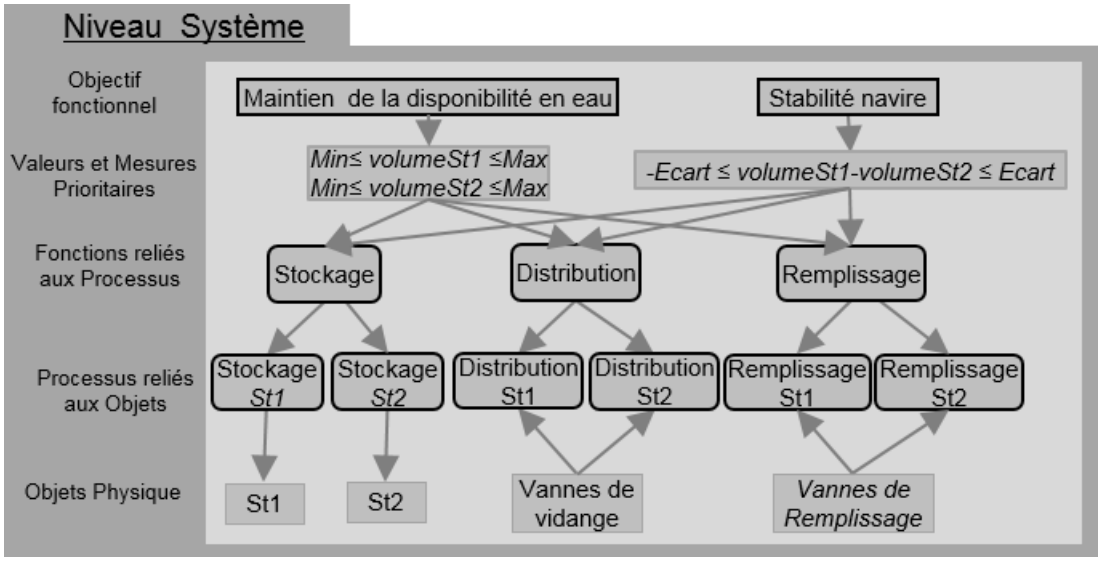


FIGURE 8.3 – Modélisation des propriétés Niveau *Système*

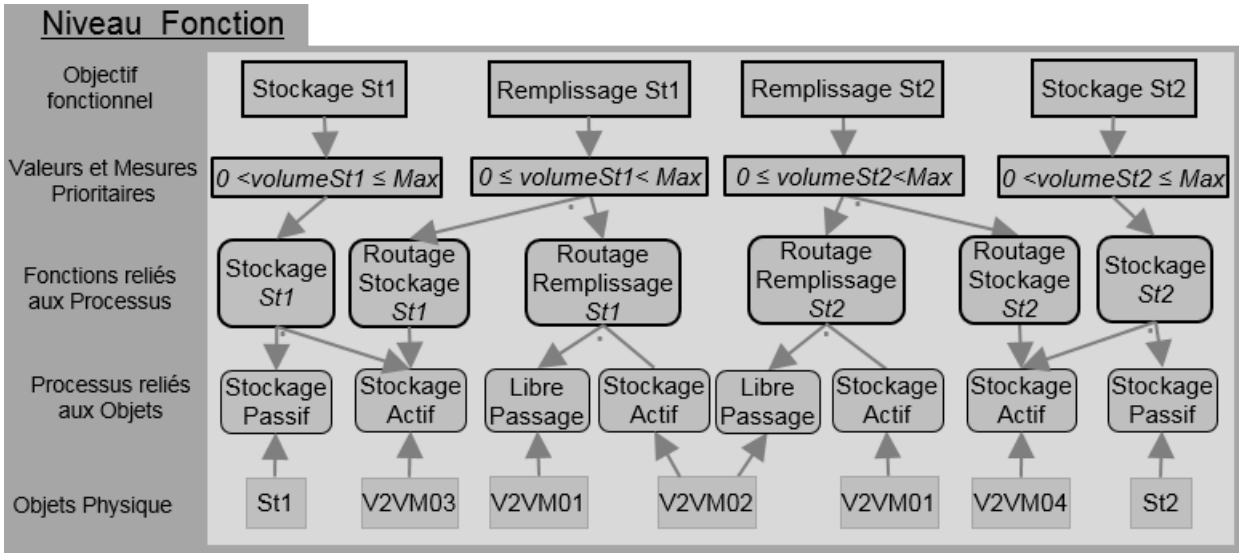


FIGURE 8.4 – Modélisation des propriétés Niveau *Fonction*

Suite à la génération du système de contrôle-commande de bas niveau de ce procédé, nous souhaitons vérifier le bon fonctionnement des alarmes liées à l'instrumentation des soutes lors d'un remplissage. Puis, nous nous intéresserons au dispositif de sécurité qui permet de stopper le remplissage lorsque la soute déborde. Il s'agira notamment de vérifier la cohérence entre le déclenchement du dispositif de sécurité (physique) et le système de gestion des alarmes.

8.1.2 Test des alarmes liées au remplissage d'une soute

Suite à la génération de bas niveau du système de contrôle-commande, nous souhaitons tester le système d'alarmes lié au remplissage d'une soute. Pour ce faire nous considérons le scénario d'exécution défini ci-dessous.

Le scénario de test 1 : Les vannes sont initialement fermées, la soute St1 contient $10m^3$ d'eau douce et la soute St2 en contient $40m^3$. Remarquant l'alarme de niveau très bas de la soute St1, l'opérateur demande l'ouverture de la vanne V2VM01 afin de la remplir. Au fur et à mesure que la soute se remplit, le transmetteur de niveau renvoie les informations liées au pourcentage de remplissage de la soute. Une fois arrivé à 98% de remplissage, l'opérateur demande la fermeture de la vanne.

A partir de ce scénario d'exécution nous pouvons définir le comportement attendu, exécuter la simulation puis interpréter les résultats (conformément à la méthodologie proposée chapitre 4).

8.1.2.1 Application de la démarche : le comportement attendu

Dérivons le comportement attendu à partir du scénario et de la modélisation des propriétés.

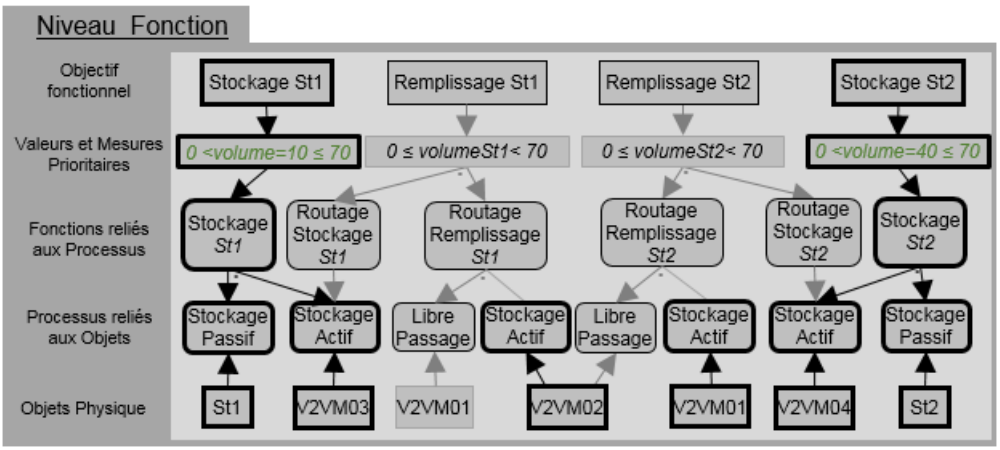


FIGURE 8.5 – Configuration initiale niveau *Fonction*

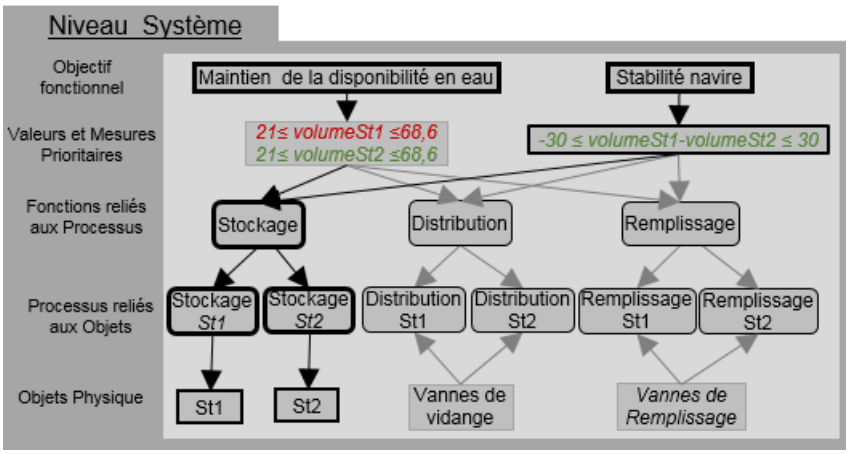


FIGURE 8.6 – Configuration initiale Niveau *Système*

Toutes les vannes étant initialement fermées, chacune d'entre elles réalise une opération de Stockage Actif. En reportant ces informations sur la modélisation des propriétés au niveau de décomposition *Fonction* (Figure 8.5), on constate qu'une configuration de Stockage St1 est implémentée ainsi qu'une configuration de Stockage St2. En reportant ces informations sur la modélisation des propriétés au niveau *Système* (Figure 8.6), nous constatons que cette configuration initiale ne permet pas d'atteindre complètement les objectifs. En effet, afin de *maintenir la disponibilité en eau*, le niveau de remplissage des soutes doit être compris entre 30% et 98% de remplissage, ce qui n'est pas le cas de la soute St1. Le niveau de remplissage de celle-ci étant par ailleurs inférieur à 15%, une alarme doit indiquer le niveau très bas de celle-ci à l'opérateur.

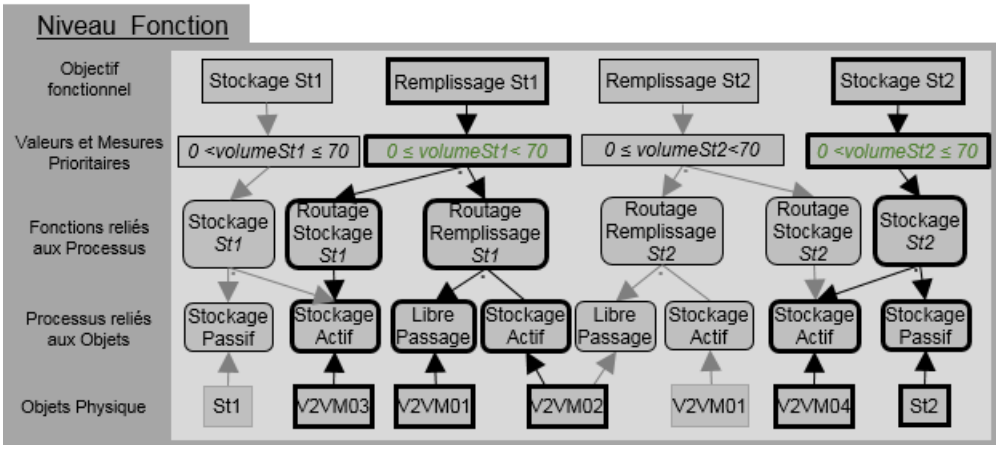


FIGURE 8.7 – Configuration attendue 1, Niveau *Fonction*

Puis, suite à la demande d'ouverture de V2VM01 de la part de l'opérateur, le procédé doit passer d'une configuration de Stockage St1 à une configuration de Remplissage St1 (Figure 8.7). Lorsque le niveau dans la soute dépasse le seuil des 15% de remplissage, l'alarme de niveau très bas doit s'éteindre, et l'alarme de niveau bas doit se déclencher. Cette dernière doit rester active tant que le seuil des 30% de remplissage n'est pas franchi. La Figure 8.8 en est une formalisation. Le critère de remplissage de St1, qui était en rouge, est maintenant en orange.

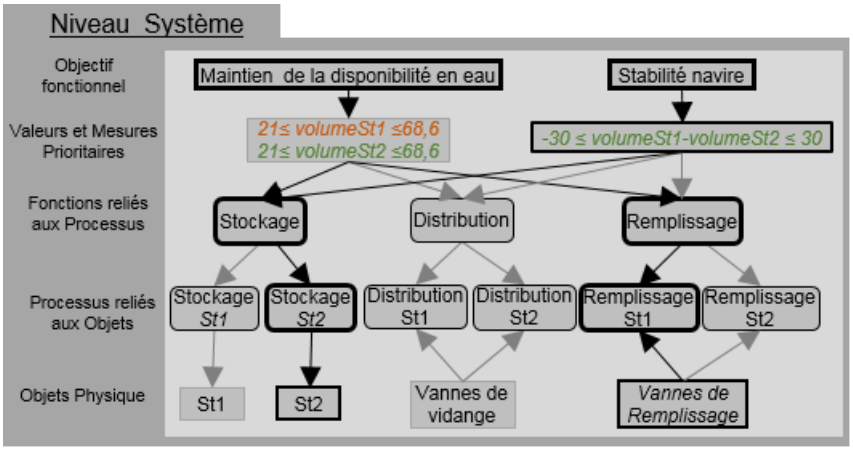


FIGURE 8.8 – Configuration attendue 2, niveau *Système*

Enfin, lorsque le seuil de 30% de remplissage de St1 est franchi, le critère de remplissage de la soute passe au vert. Le système passe alors dans un état qui permet d'atteindre les objectifs (Configuration attendue 3). L'opérateur n'effectuant pas de requête, le procédé reste dans sa configuration de Remplissage St1 et Stockage St2. Par conséquent, le niveau de remplissage de la soute continue d'augmenter, jusqu'à atteindre le seuil des 98% de remplissage. A cet instant, le niveau de remplissage haut est atteint, ce qui doit déclencher l'alarme de niveau haut (Configuration attendue 4).

L'opérateur, en voyant l'alarme, demande la fermeture de V2VM01 pour mettre un terme au remplissage de la soute. Le procédé doit alors se retrouver dans la configuration de Stockage St1 et Stockage St2 (Figure 8.9).

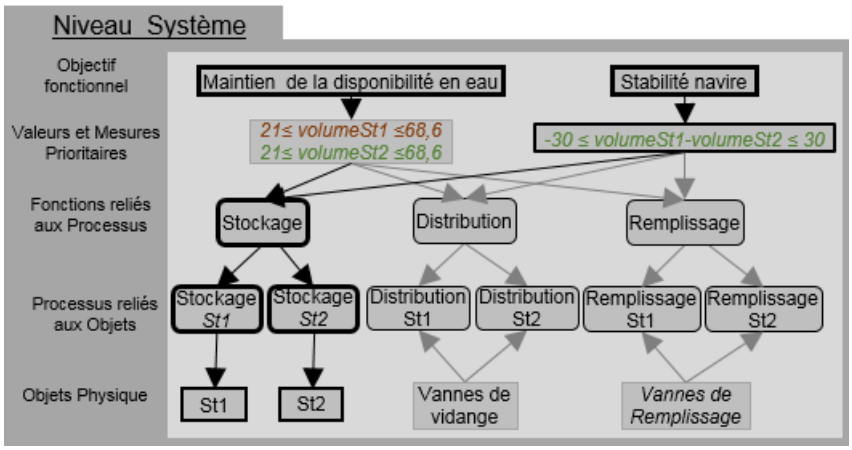


FIGURE 8.9 – Configuration attendue finale, niveau *Système*

Le comportement attendu étant défini, afin de pouvoir lancer la simulation il nous faut disposer du modèle de simulation du procédé. En appliquant le *flot de simulation* (cf chapitre 7), en moins de 10 secondes nous obtenons le modèle de simulation et la librairie associée. L'exécution de la simulation peut maintenant être lancée.

8.1.2.2 Exécution de la simulation

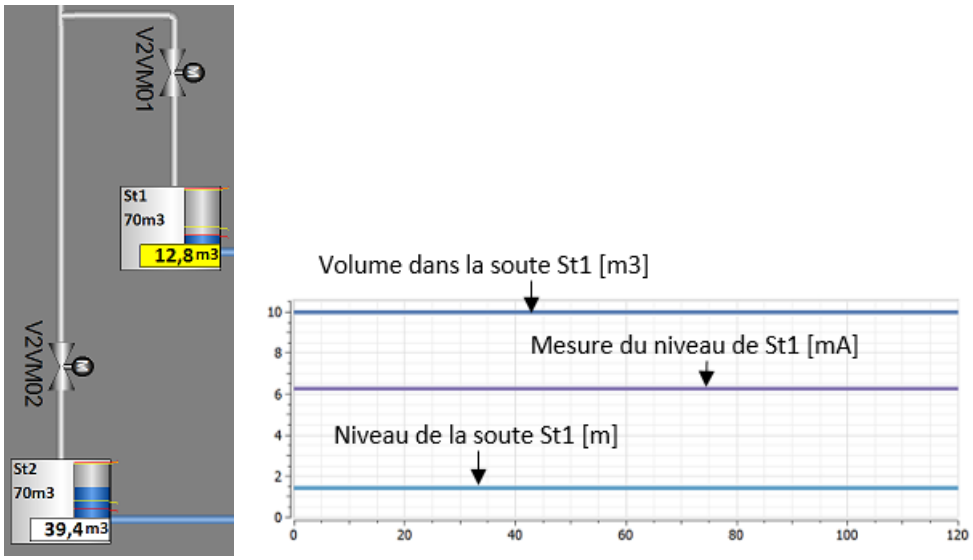
Pour exécuter la simulation, nous importons dans le logiciel SystemModeler la librairie générée ainsi que le modèle de simulation. Le modèle de simulation qui comporte 298 équations et 298 variables est compilé en moins d'une minute, de même pour la génération de l'exécutable.

Afin d'apporter une preuve de concept, nous nous plaçons du point de vue de l'automaticien qui souhaite vérifier et corriger, de manière itérative, son programme en amont de la phase d'implantation.

8.1.2.2.1. Détection d'une erreur dans le programme de commande

Au lancement de la simulation, l'interface de supervision indique à l'opérateur que le niveau dans la soute St1 est bas (via une alarme de niveau bas). Ne s'agissant pas d'une

alarme de niveau très bas dans la soute, l'opérateur ne demande pas l'ouverture de la vanne V2VM01 celle-ci étant conditionnée par l'apparition d'une alarme de niveau très bas (dans le scénario).



(a) IHM supervision (b) Résultats simulation

FIGURE 8.10 – Aperçu de l'IHM de supervision et du procédé simulé

De plus, l'interface de supervision indique que la soute St1 contient 12,8m^3 et 39,4m^3 pour la soute St2 (Figure 8.10a), tandis que les résultats de la simulation indiquent que St1 contient 10m^3 (Figure 8.10b) et 40m^3 pour St2. Nous constatons ainsi que les volumes affichés des soutes ne correspondent pas à ceux définis, mais néanmoins que l'alarme de niveau bas est cohérente avec le volume affiché. L'erreur ne semble donc pas se situer au niveau de l'IHM de supervision.

En regardant du coté du programme de commande (Figure 8.11), on s'aperçoit que les valeurs affichées sur l'IHM correspondent à celles envoyées par le programme de commande. Il en va de même pour l'alarme. Pourtant le signal de mesure du transmetteur de niveau (envoyé et reçu) est bien de 6,285714 mA, ce qui correspond bien à un volume de 10m^3. Il y a donc un problème dans le traitement du signal de mesure au niveau du programme de commande.

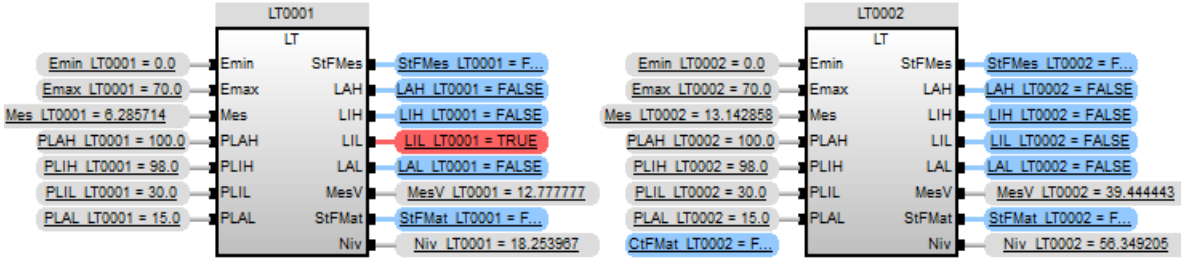


FIGURE 8.11 – Extrait de l'état initial du programme de commande

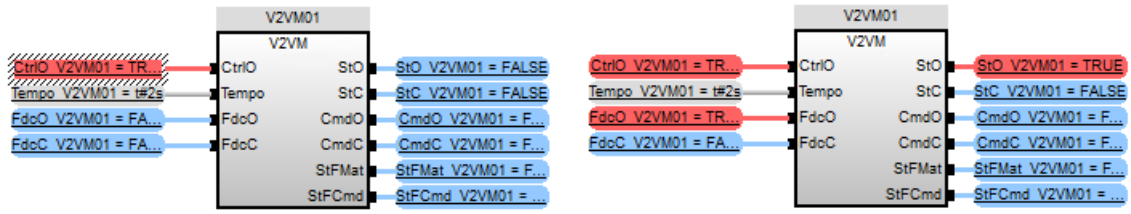
Ainsi, la mise en oeuvre de ce test, nous a permis de nous apercevoir qu'il y avait une erreur de conversion dans le programme de commande. Celle-ci n'avait jusqu'alors pas été détectée.

Après correction du programme de commande, nous pouvons recommencer le test afin de valider le système d'alarmes.

8.1.2.2.2. Validation du système d'alarmes

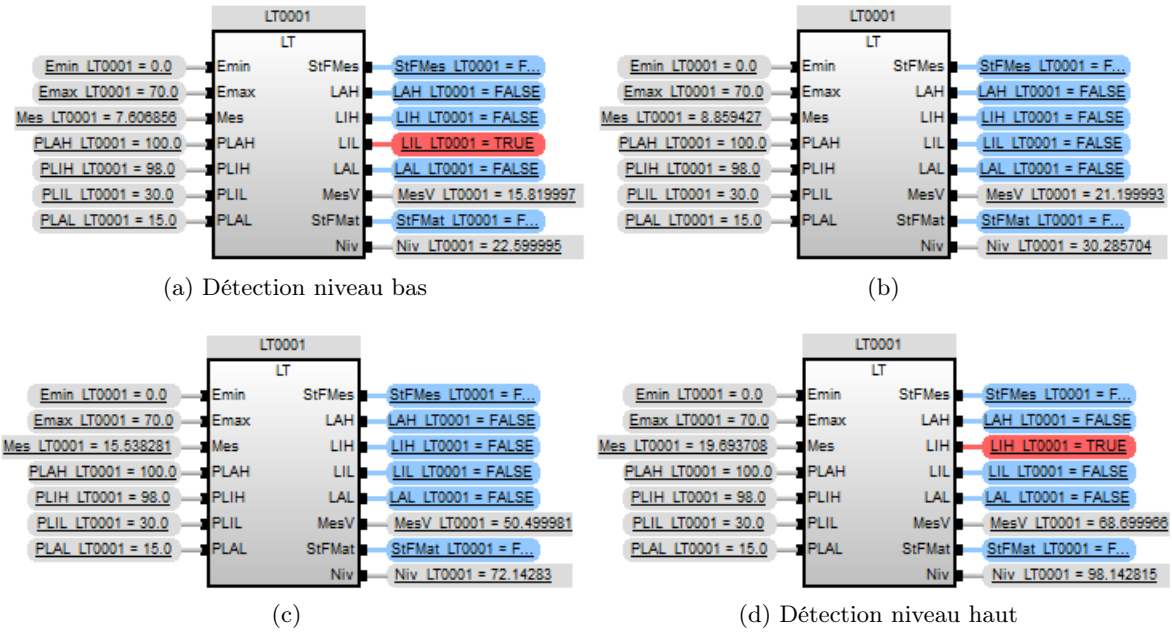
Une capture d'écran de l'état initial du programme de commande est donnée en Annexe D. Toutes les vannes apparaissent en position fermée (`FdcC = TRUE`). La mesure du niveau dans la soute St1 indique qu'elle est à 14% de remplissage et qu'elle contient 10m^3 d'eau. La mesure du niveau de la soute St2 indique qu'elle est à 57% de remplissage et qu'elle contient 40m^3 d'eau. D'autre part, une alarme est active : il s'agit du niveau très bas dans la soute St1.

Remarquant cette alarme, l'opérateur demande l'ouverture de la vanne V2VM01. La variable CtrlO_V2VM01 passe alors à `TRUE`. Suite à l'envoi de la commande d'ouverture, la vanne s'ouvre : FdcC passe à `FALSE` (Figure 8.12a), puis FdcO passe à `TRUE` (Figure 8.12b). La vanne est ainsi en position ouverte. On voit le niveau dans la soute St1 augmenter, via la mesure du capteur LT0001 (Figure 8.13).



(a) Manoeuvre de la vanne (b) Vanne ouverte

FIGURE 8.12 – Ouverture de la vanne V2VM01



(a) Détection niveau bas (b)

(c) (d) Détection niveau haut

FIGURE 8.13 – Évolution du niveau dans la soute

Une fois le seuil des 15% de remplissage franchi, l'alarme de niveau très bas s'éteint tandis que celle de niveau bas s'active. Cette dernière reste active tant que les 30% de remplissage ne sont pas dépassés (cf Figure 8.13a et Figure 8.13b). Une fois les 98% de remplissage atteints, l'alarme de niveau haut s'active (Figure 8.13d). Remarquant l'alarme de niveau haut, l'opérateur demande la fermeture de la vanne V2VM01. Ayant reçu la commande de fermeture, la vanne se ferme. La simulation s'achève une fois la vanne en position fermée.

Une partie des résultats de la simulation est représentée sur la Figure 8.14. Il s'agit de la courbe représentant le niveau dans la soute St1, et de la courbe représentant le signal de mesure envoyé à la commande. On peut voir que les 15% de remplissage (correspondant à un niveau de 1,5m dans la soute) sont atteints à t=34,2s, les 30% sont atteints à t=86,7s, et enfin que les 98% sont atteints à t=324.7s. Puis à partir de t=329,6s, le niveau dans la soute est constant (9,94m).

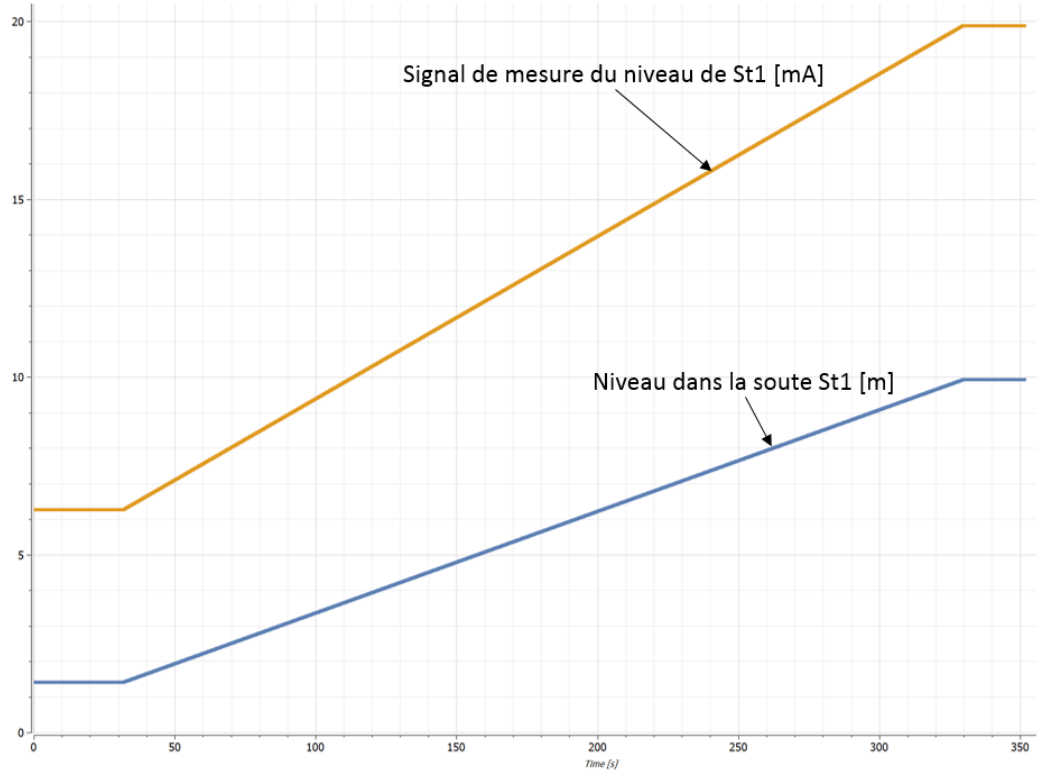


FIGURE 8.14 – Evolution du niveau dans la soute St1

L'évolution de l'état de la vanne V2VM01 est représentée sur la Figure 8.15. Ainsi, à t=31,7s la vanne quitte sa position fermée, entamant sa phase d'ouverture. Puis à t=329,6s elle se retrouve de nouveau en position fermée.

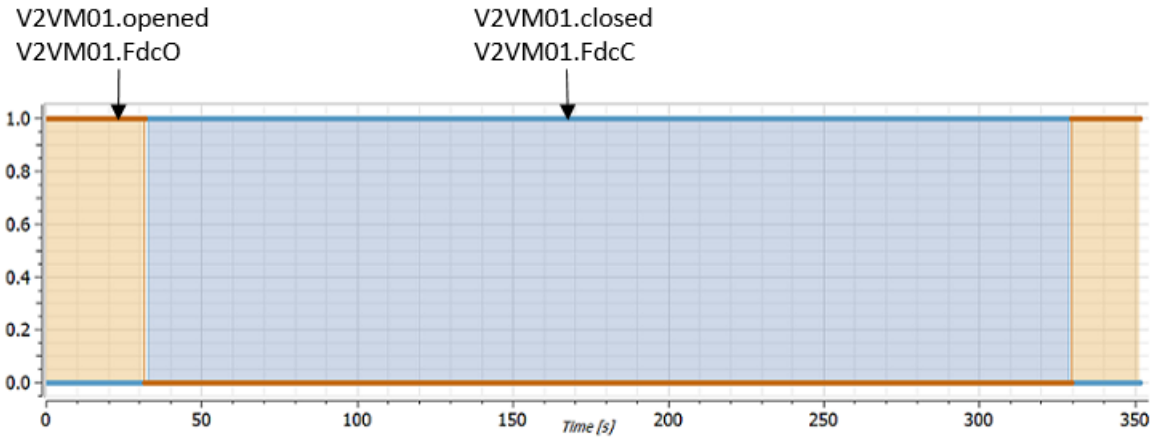


FIGURE 8.15 – Evolution de l'état de V2VM01

8.1.2.3 Application de la démarche : analyse des résultats de la 2eme exécution de la simulation

Analysons les résultats de la simulation en appliquant la démarche (proposée au chapitre 4). En reportant les résultats issus de la simulation sur la modélisation des propriétés, nous devons obtenir les configurations attendues.

Configuration initiale attendue entre t=0s et t=31s

Au démarrage de la simulation toutes les vannes sont en position fermée (`closed=TRUE`), celles-ci réalisent donc des opérations de Stockage Actif. La configuration obtenue aux instants compris entre t=0s et t=31s est représentée sur la Figure 8.16. Celle-ci correspond à la configuration initiale attendue (cf Figure 8.5).

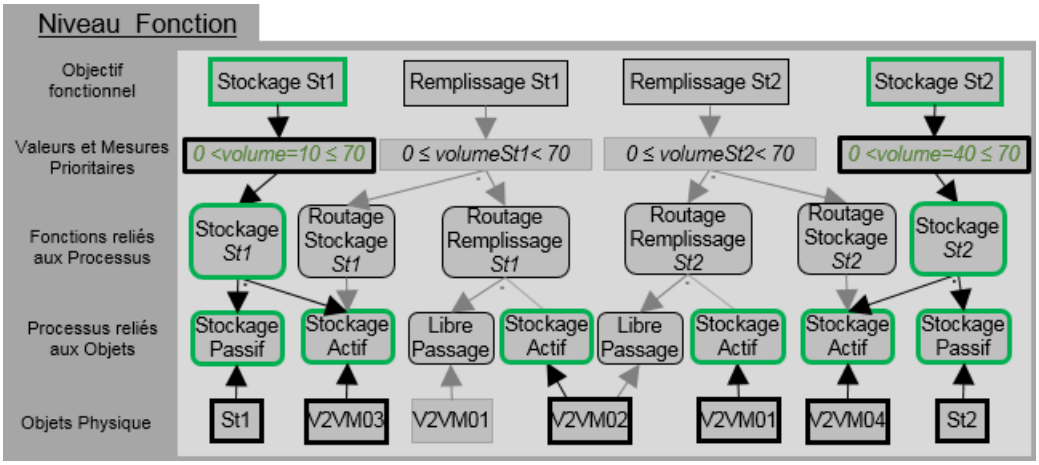


FIGURE 8.16 – Configuration initiale obtenue

Configuration de Remplissage St1 attendue entre t=32,6 et t=325s

Suite à l'envoi de la commande d'ouverture, la vanne V2VM01 quitte sa position fermée à t=31,7s et atteint sa position ouverte à t=32,6s. Ainsi à partir de t=32,6s une opération de Libre Passage est réalisée par V2VM01. Une configuration de Remplissage St1 est donc implémentée. La Figure 8.17 représente la modélisation du comportement obtenu à t=33s, au Niveau *Fonction*. La seule différence avec celle obtenue à t=32,6s étant le volume dans la

soute qui a augmenté. Ceci est tout à fait normal puisqu'une configuration de la fonction de Remplissage St1 est implémentée. Si l'on s'intéresse maintenant à la modélisation au Niveau *Système* (Figure 8.18), les 15% de remplissage de la soute St1 n'ayant pas été atteints, son niveau est toujours très bas, l'alarme associée est en conséquence toujours active. Il s'agit bien de la Configuration attendue 1.

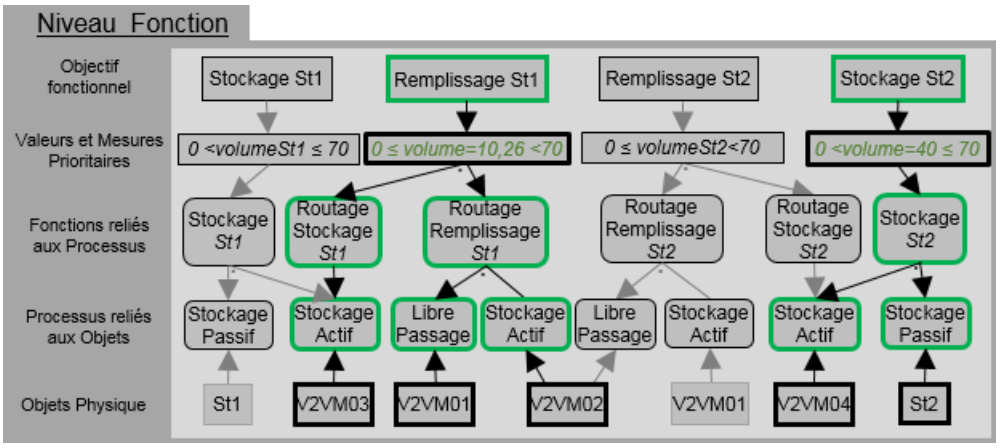


FIGURE 8.17 – Configuration obtenue à t=33s, Niveau *Fonction*

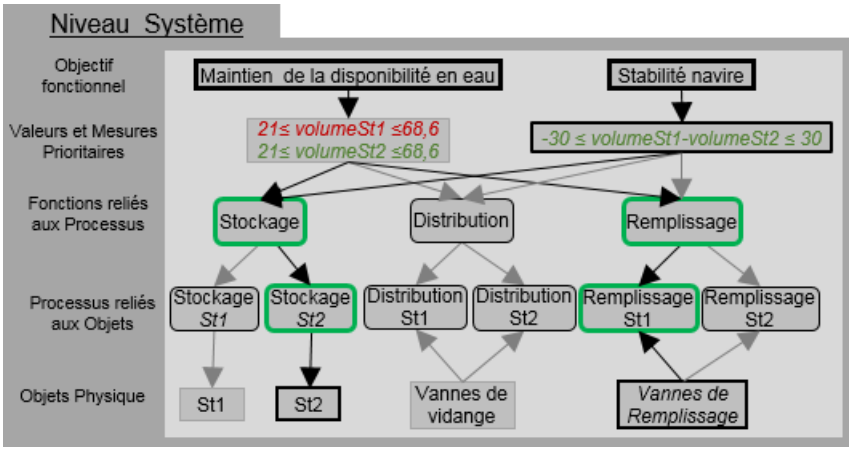


FIGURE 8.18 – Configuration obtenue à t=33s, Niveau *Système*

Entre les instants t=34,2s et t=86,7s, le niveau de remplissage est bas (entre 15% et 30% de remplissage). La configuration obtenue entre ces instants correspond à celle de la Figure 8.19 (outre la valeur du volume dans la soute St1 qui évolue) et celle Figure 8.20. Ainsi, lorsque le volume de la soute St1 est de 15,8m^3 (à t=60,7s), conformément à la modélisation Niveau *Système*, l'alarme de niveau très bas a laissé sa place à l'alarme de niveau bas (cf Figure 8.12a).

Entre t=86,7s et t=324,7s le volume dans la soute St1 est compris entre 21m^3 et 68,6m^3. Le critère concernant le maintient de la disponibilité en eau est alors assuré : le procédé se trouve dans un état qui lui permet d'atteindre ses objectifs (Configuration attendue 3).

Puis, une fois le seuil des 98% de remplissage atteint celui-ci se trouve dans un état de niveau haut de la soute St1 (Figure 8.21), dont l'opérateur est averti par une alarme de niveau haut. Ceci correspond à la Configuration attendue 4.

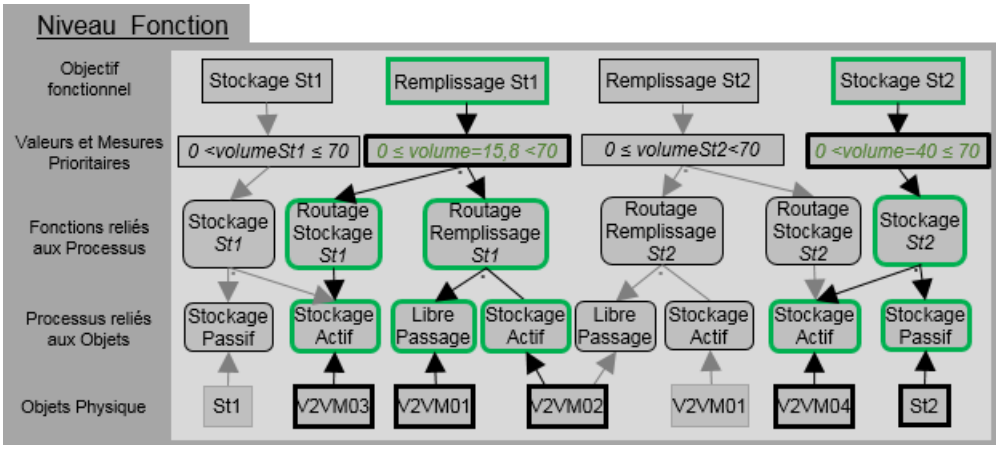


FIGURE 8.19 – Configuration obtenue à t=60,7s, Niveau *Fonction*

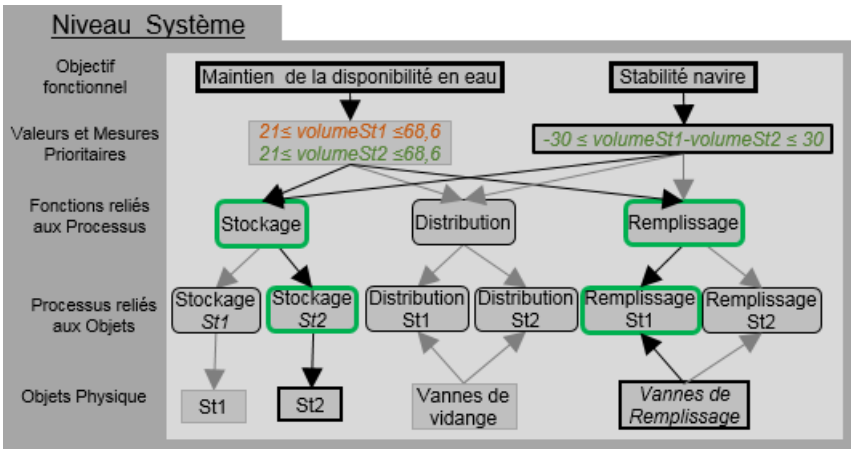


FIGURE 8.20 – Configuration obtenue à t=60,7s, Niveau *Système*

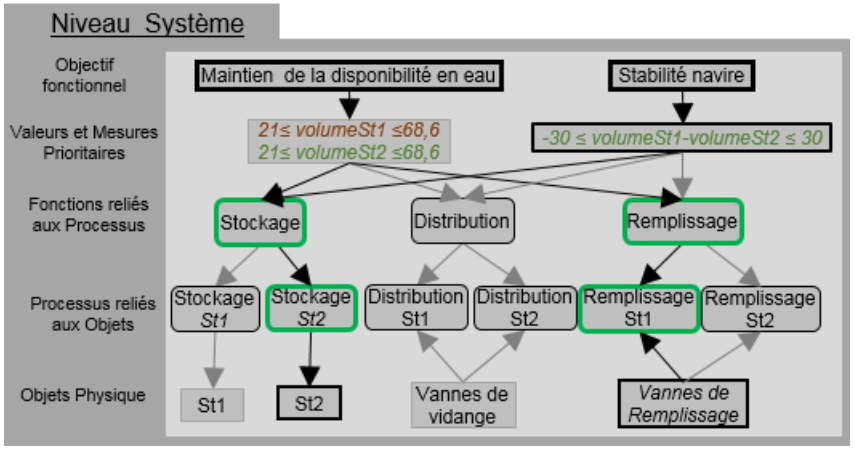


FIGURE 8.21 – Configuration obtenue à t=325s, Niveau *Système*

Configuration finale attendue à partir de t=329.6s

Voyant l'alarme de niveau haut, l'opérateur a demandé la fermeture de la vanne V2VM01, laquelle atteint sa position fermée à t=329.6s. Dès lors ce n'est plus une opération de Libre

Passage qui est réalisée par celle-ci mais une opération de Stockage Actif. A partir de cet instant le procédé se retrouve dans une configuration de Stockage St1 et Stockage St2 (Figure 8.22). Il s'agit de la configuration finale attendue.

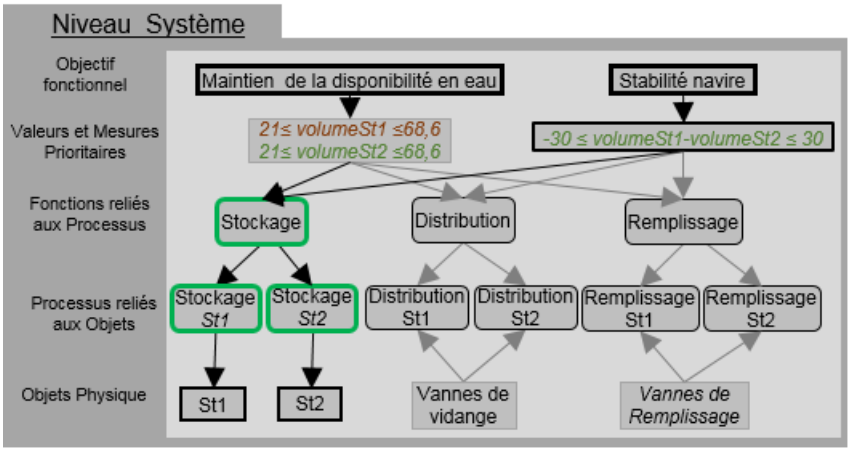


FIGURE 8.22 – Configuration obtenue à partir de t=329,6s, Niveau *Système*

En conclusion, cette deuxième exécution de la simulation nous permet de valider le système d'alarmes. De plus nous pouvons constater que le problème de traitement du signal de mesure de niveau par le programme de commande a bien été corrigé. En effet, cette fois-ci nous avons pu observer la correspondance des valeurs des volumes des soutes du programme de commande avec celles du procédé simulé.

8.1.3 Tests de cohérence avec le dispositif de sécurité

Nous souhaitons nous assurer de la cohérence du système d'alarmes lors du déclenchement d'un dispositif de sécurité du procédé. Le dispositif de sécurité de notre cas d'étude, consiste à fermer la vanne de remplissage d'une soute lorsque celle-ci est déjà pleine, via un signal électrique. Il s'agit donc de s'assurer, entre autres, qu'il n'y a pas d'alarme de défaut de commande lorsque le dispositif de sécurité est enclenché. Pour ce faire, nous avons défini le scénario de test ci-dessous.

Le scénario de test 2 : Les vannes sont initialement fermées, la soute St1 contient 40m^3 d'eau douce et la soute St2 en contient 40m^3. L'opérateur demande l'ouverture de la vanne V2VM02. La soute se remplit et l'opérateur, occupé, ne voit pas les alarmes indiquant le niveau haut et très haut de la soute. Une fois le seuil de trop plein détecté par LS0002, le dispositif de sécurité doit alors s'enclencher en envoyant une commande électrique à la vanne pour qu'elle se ferme.

A partir de ce scénario nous définissons le comportement attendu, puis nous analyserons les résultats de la simulation suite à son exécution (conformément à notre méthodologie de vérification).

8.1.3.1 Application de la démarche : le comportement attendu

Initialement toutes les vannes étant fermées, une opération de Stockage Actif est réalisée par chacune d'elles. A partir de la modélisation au niveau *Fonction*, nous en déduisons que le procédé est dans une configuration de Stockage St1 et Stockage St2 (Figure 8.23).

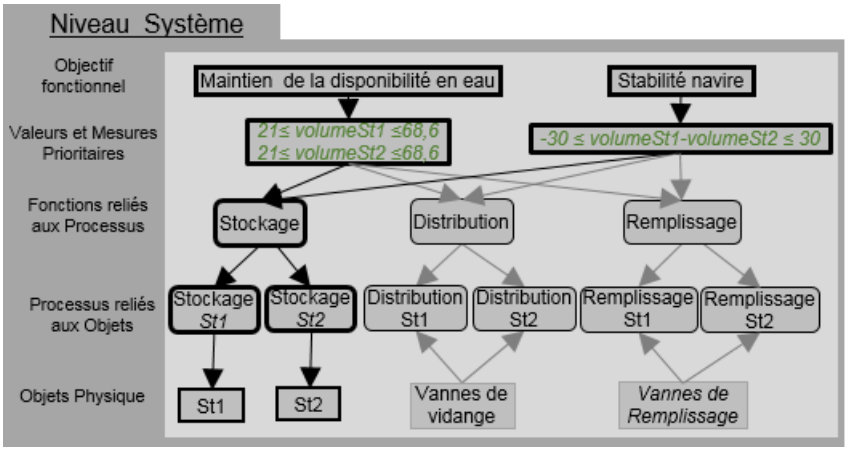


FIGURE 8.23 – Configuration initiale niveau *Système*

Suite à la demande d'ouverture de la vanne V2VM02, une opération de Libre Passage doit être réalisée par celle-ci. Le procédé doit ainsi passer de l'implémentation d'une configuration de Stockage St2 à l'implémentation d'une configuration de Remplissage St2 (Figure 8.24 et Figure 8.25). Tant que le niveau dans la soute n'a pas dépassé les 98% de remplissage, le procédé reste dans cette configuration (Configuration attendue 1). Lorsqu'il les dépasse, bien que le procédé soit toujours dans une configuration de Remplissage St2, le critère associé au maintient de la disponibilité en eau n'est plus respecté (niveau haut de St2).

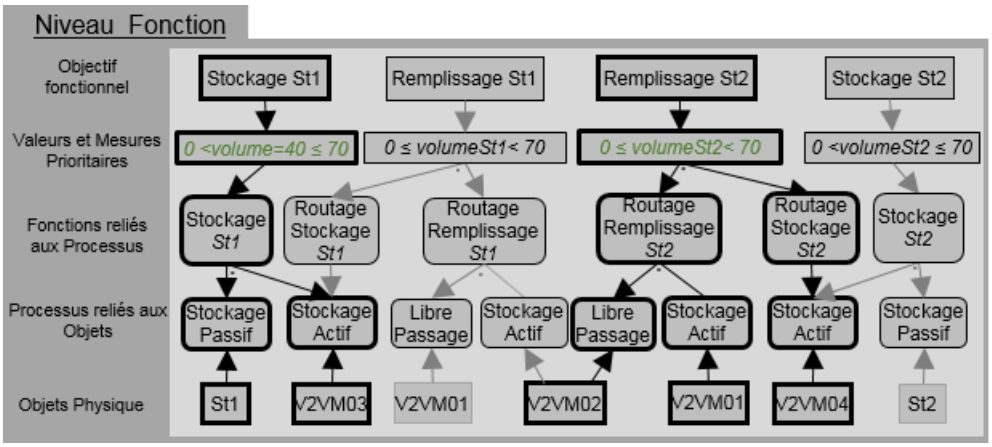


FIGURE 8.24 – Configuration attendue 1, niveau *Fonction*

Une fois la soute totalement remplie (100% de remplissage), l'opérateur n'ayant pas effectué de requête, le procédé est toujours dans une configuration de Remplissage St2. Le seuil de trop-plein est atteint, enclenchant le dispositif de sécurité (fermeture de la vanne V2VM02). Le procédé doit alors se retrouver dans une configuration de Stockage (Stockage St1 et Stockage St2) comme représenté Figure 8.26.

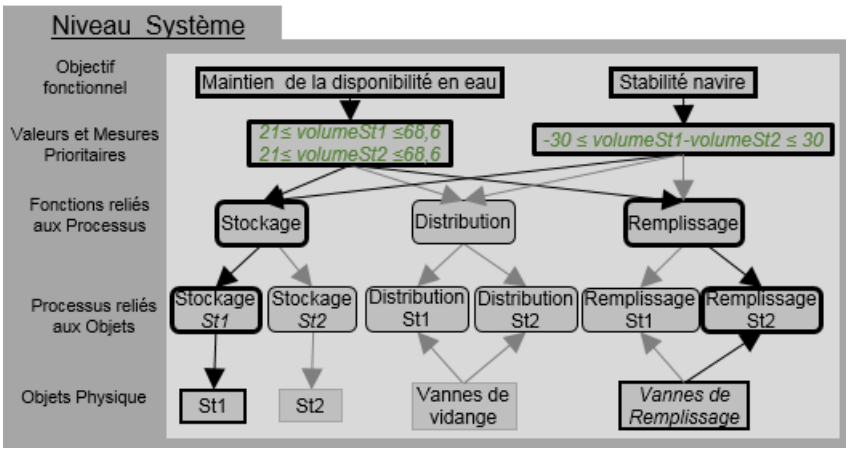


FIGURE 8.25 – Configuration attendue 1, niveau *Système*

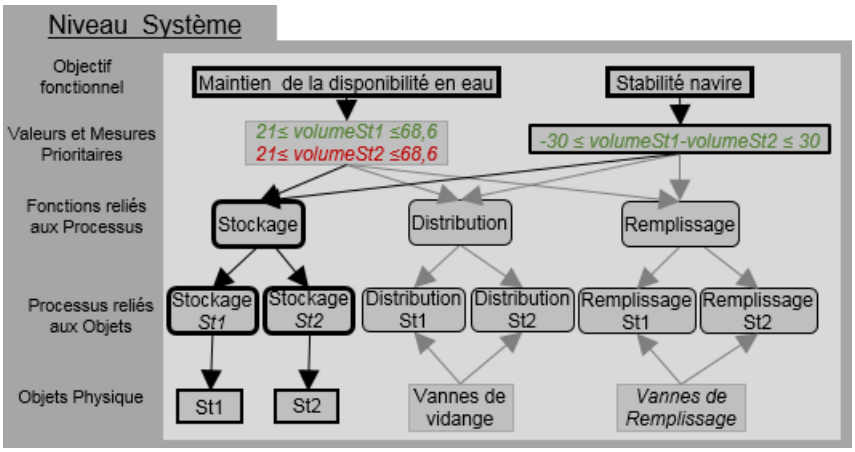


FIGURE 8.26 – Configuration finale, niveau *Système*

8.1.3.2 Exécution de la simulation

Au démarrage de la simulation, l'IHM de supervision et le programme de commande indiquent que les soutes contiennent 40m^3, et que toutes les vannes sont fermées. L'opérateur demande l'ouverture de V2VM02 via l'interface de supervision. Suite à l'ouverture de la vanne, le niveau dans la soute St2 augmente. L'alarme de niveau haut s'affiche sur l'IHM (image du milieu Figure 8.27) après que le seuil des 98% de remplissage a été dépassé. Puis, lorsque les 100% de remplissage sont atteints l'alarme de niveau haut laisse la place à celle de niveau très haut. Cependant, l'opérateur ne remarque pas les alarmes, il ne demande donc pas l'arrêt du remplissage. L'alarme de trop-plein s'active. La vanne V2VM02 se ferme, il apparait alors une alarme de défaut de commande (image de droite Figure 8.27).

Dans le programme commande (Figure 8.28), la vanne V2VM02 y apparait comme étant en position fermée (`StC=TRUE`) et l'alarme de défaut de commande est active (`StFCmd=TRUE`). Deux autres alarmes sont également actives : celle de niveau très haut dans la soute St2 (`LAH_LT0002=TRUE`), et celle du trop-plein (`LAH_LS0002=TRUE`).

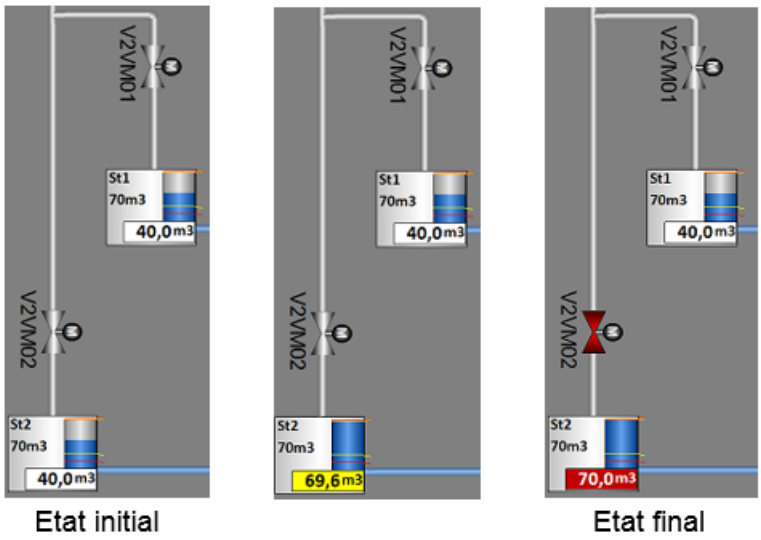


FIGURE 8.27 – Capture d'écran de l'IHM durant la simulation

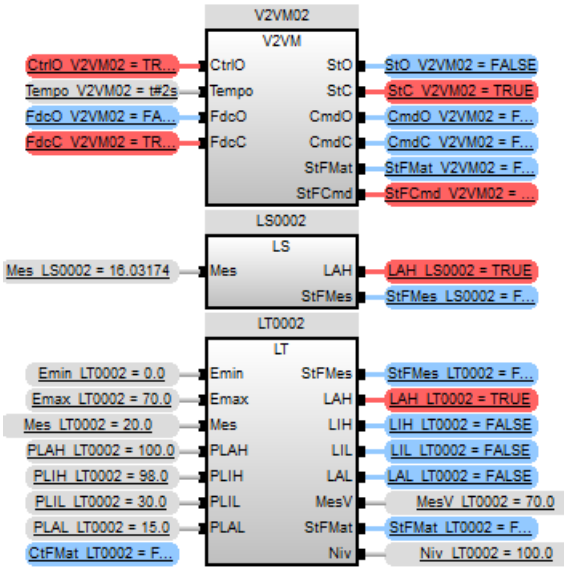


FIGURE 8.28 – Programme de commande après enclenchement du dispositif de sécurité

Un extrait des résultats de simulation est présenté Figure 8.29. La courbe orange représente l'évolution de la mesure de niveau dans la soute St2. La courbe en bleu représente le signal de mesure du détecteur de trop plein. Lorsque la sonde du capteur est dans l'air, elle renvoie un signal de 8,1mA tandis que lorsqu'elle est dans l'eau elle renvoie un signal de 16,1mA. Enfin en vert est représenté le dispositif de sécurité qui commande la fermeture de la vanne V2VM02, dès que le trop-plein est détecté.

L'évolution de la position de la vanne V2VM02 est représentée Figure 8.30. La vanne quitte sa position fermée à t=46,7s, et atteint sa position ouverte à t=47,6s. Celle-ci reste en position ouverte jusqu'à la réception de la commande électrique de fermeture à t=196,71s. Ainsi, à t=197,8s la vanne V2VM02 se retrouve à nouveau en position fermée.

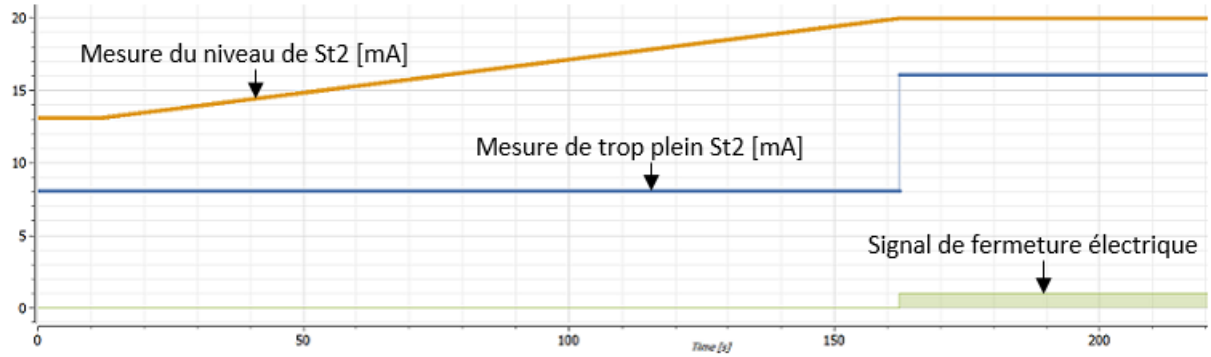


FIGURE 8.29 – Évolution des signaux de l'instrumentation de la soute St2

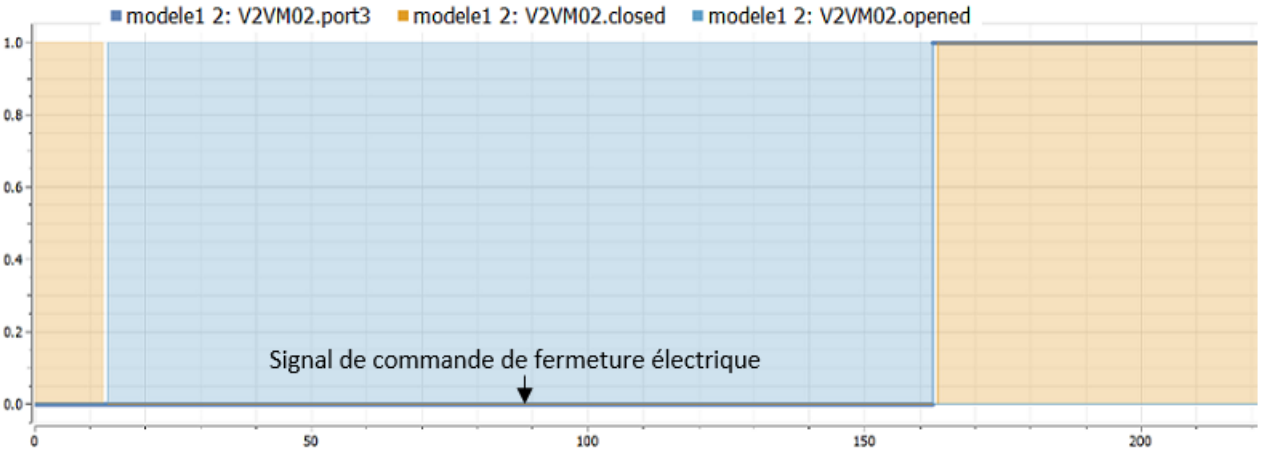


FIGURE 8.30 – Évolution de la position de la vanne

8.1.3.3 Application de la démarche : analyse des résultats de la simulation

Appliquons maintenant la démarche pour l'analyse des résultats de la simulation.

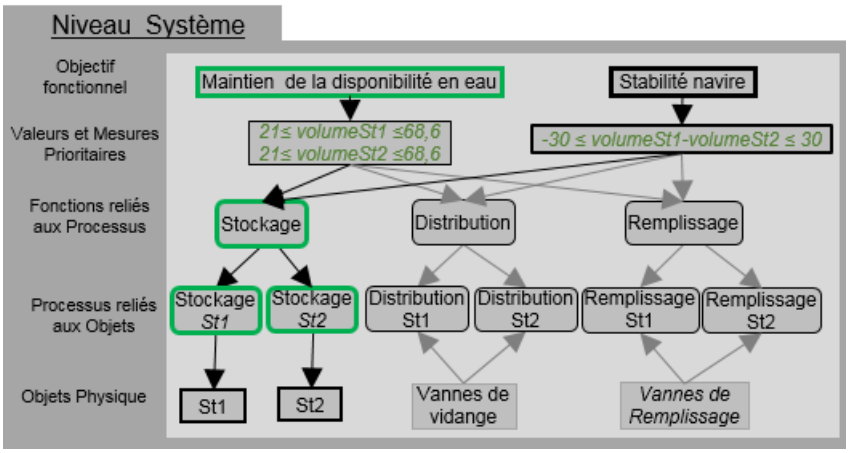


FIGURE 8.31 – configuration initiale obtenue, Niveau *Système*

Au démarrage de la simulation toutes les vannes sont en position fermée, chacune d'entre elles implémentant une opération de Stockage Actif. Le procédé se trouve donc dans une

configuration de Stockage St1 et Stockage St2 (Figure 8.31). Le procédé reste dans cette configuration jusqu'à la réception de la commande d'ouverture de V2VM02.

Le procédé, qui est dans une configuration de Stockage St1 et Stockage St2 entre les instants t=0s et t=46,7s, se retrouve à partir de t=47,6s dans une configuration de Stockage St1 et de Remplissage St2. En effet, à partir de t=47,6s V2VM02 est en position ouverte, celle-ci réalise alors une opération de Libre Passage. Ainsi, tant que le remplissage de St2 est inférieur à 98% le procédé se trouve dans la configuration attendue 1 (Figure 8.25).

A t=189,7s les 98% de remplissage de St2 sont atteints. Bien que le procédé soit toujours dans une configuration de Stockage St1 et de Remplissage St2, le critère lié au maintient de la disponibilité en eau n'est plus assuré. Ceci est indiqué à l'opérateur via l'alarme de niveau haut, mais aucune action n'est engagée de sa part. Ainsi, à t=194,7s le système est toujours dans la même configuration (Figure 8.32).

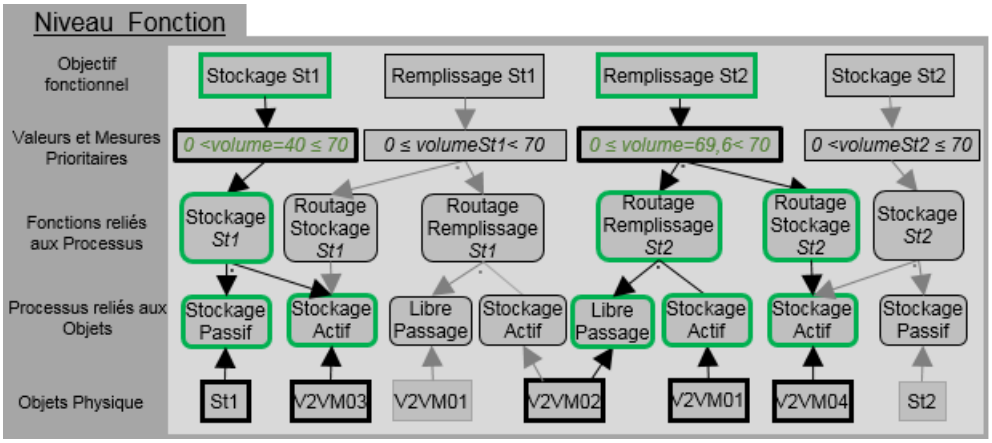


FIGURE 8.32 – Configuration obtenue à t=194,7s, Niveau *Fonction*

Le remplissage de la soute se poursuit, et à t=196,7s celle-ci est totalement remplie, l'alarme de niveau très haut se déclenche. Néanmoins l'opérateur n'ayant toujours pas réagi, à t=196,71s, le seuil de trop plein est atteint et l'alarme de trop plein s'active. A cet instant, le dispositif de sécurité se déclenche : un signal de commande électrique est envoyé à la vanne de remplissage de la soute.

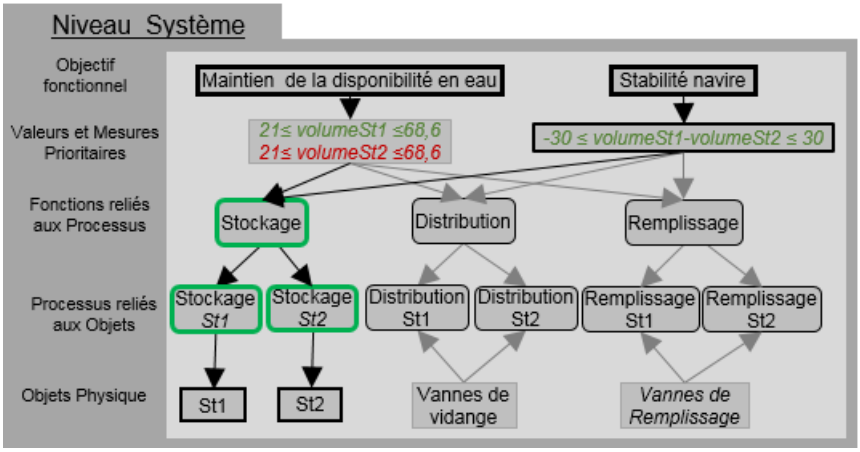


FIGURE 8.33 – Configuration obtenue à t=197,8s, Niveau *Système*

A t=196,9s celle-ci quitte sa position ouverte et elle atteint sa position fermée à t=197,8s. Une opération de Stockage Actif est alors réalisée par V2VM02. A partir de cet instant, ce n'est donc plus une configuration de Remplissage St2 qui est implémentée mais une configuration de Stockage St2 (Figure 8.33).

En conclusion, conformément au comportement attendu le procédé reste dans une configuration de Stockage St1 et de Remplissage St2 jusqu'à ce que le dispositif de sécurité s'enclenche (c'est à dire entre t=47,6s et t=196,9s). Le déclenchement du dispositif de sécurité entraîne une reconfiguration et le procédé se retrouve à nouveau dans une configuration de Stockage St1 et Stockage St2.

Toutefois, bien que cette reconfiguration soit requise par la situation, une alarme de défaut de commande est générée par le programme de commande. Il apparait ainsi une incohérence entre le système de contrôle-commande et le dispositif de sécurité : force est de constater que le système de contrôle-commande n'a pas détecté que la fermeture de V2VM02 était due au dispositif de sécurité, et qu'il n'y avait donc pas de "défaut". Nous avons donc été en mesure de détecter une incohérence du contrôle-commande et un changement de mode de fonctionnement (passage en mode de sécurité) du système, suite à une reconfiguration.

8.1.4 Bilan

Nous venons d'appliquer notre démarche de vérification sur un cas d'école. Suite à la génération de bas-niveau du système de contrôle-commande, nous avons mis en oeuvre une série de tests afin de vérifier le bon fonctionnement du système d'alarmes. En appliquant notre méthodologie pour interpréter les résultats des tests, nous avons été capables de mettre en lumière des dysfonctionnements, qui ont par la suite pu être corrigés.

En outre, le formalisme de modélisation des propriétés multi-niveau permet d'avoir une synthèse de la connaissance, contextualisée, et ainsi de facilement comparer le comportement attendu au comportement obtenu. Ceci permet de mener des tests dès les premières phases de conception pour garantir la cohérence sémantique. Lors de la mise en oeuvre de vérifications du système d'alarmes lié au remplissage d'une soute, nous avons ainsi détecté une erreur dans le programme de commande bien que celle-ci ne fasse pas l'objet de la vérification. Il s'agit d'une erreur dans la conversion entre la valeur de l'intensité du signal de la mesure et la valeur de la grandeur mesurée (le niveau dans la soute). Nous avons également cherché à tester la cohérence entre dispositif de sécurité et le système d'alarmes. Cela nous a permis de voir que le programme de contrôle-commande n'avait pas conscience que le dispositif de sécurité était enclenché, mettant ainsi en évidence les limites de la détection locale de défauts (uniquement au niveau d'un élément). Ce dernier point est d'autant plus intéressant qu'il semble possible, dès les premières phases de conception, de s'interroger sur la cohérence des différents modèles du contrôle-commande avec les configurations des modes de fonctionnement du procédé (mode manuel, automatique, semi-automatique, de sécurité ...). L'application de notre démarche de vérification apporte ainsi un support pour identifier et localiser des incohérences, même lorsque ces dernières ne sont pas l'objet principal de la vérification.

Forts de ces résultats sur un cas d'école, il s'agit maintenant de considérer un cas réel en vue d'apporter une preuve d'usage de nos propositions, et en particulier par rapport à l'obtention du modèle de simulation.

8.2 Passage à l'échelle et conclusion

8.2.1 Passage à l'échelle

Considérons le système d'eau douce sanitaire suivant (Figure 8.34). Ce dernier est similaire au système considéré chapitre 4. Chaque soute est reliée à chacune des pompes (au nombre de trois) par une vanne distincte. Ainsi, à chaque pompe sont connectées deux vannes d'aspiration : une des vannes de vidange de la soute St1, et une de celles de la soute St2.

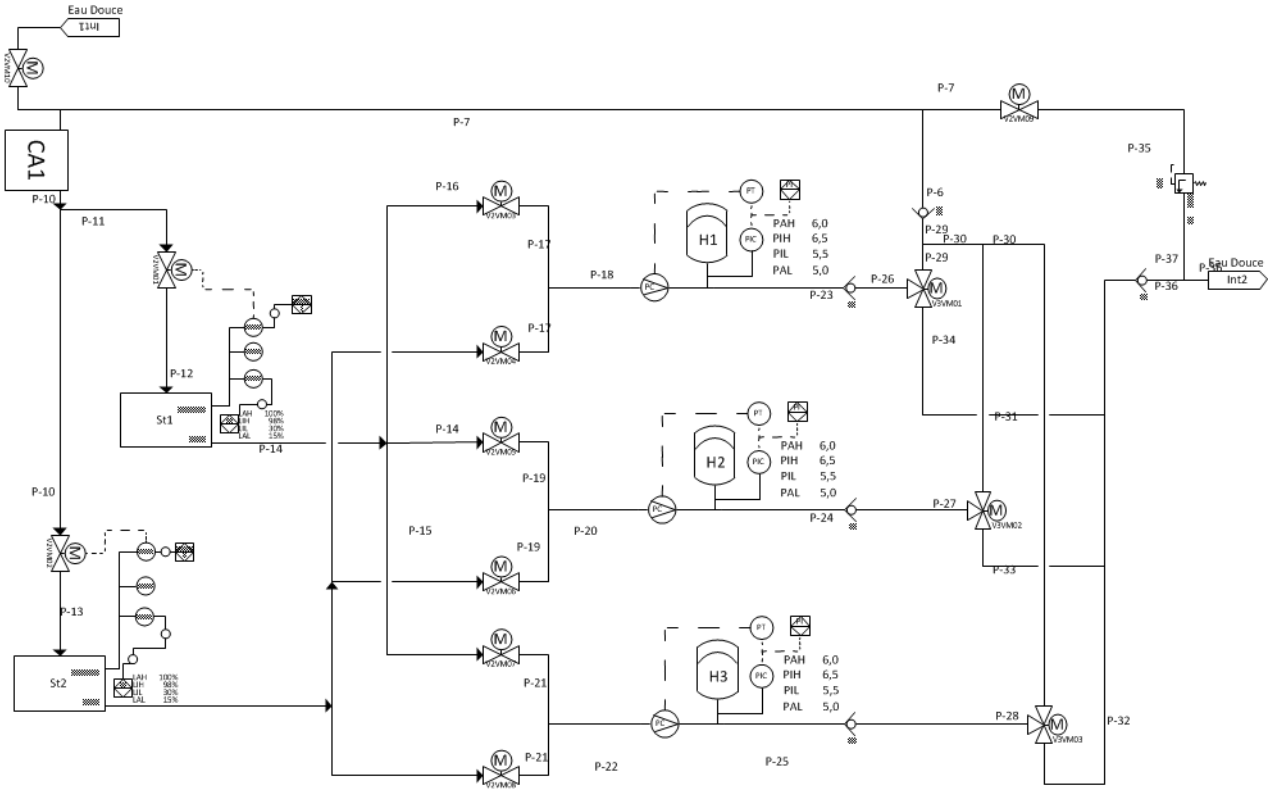


FIGURE 8.34 – Système d'eau douce sanitaire

Suite à l'obtention du système de contrôle-commande de bas niveau, nous souhaitons tester la possibilité de réaliser les fonctions de haut niveau (un transfert entre les soutes), avant l'étape de spécification des fonctions de haut niveau. Nous considérerons à ce titre le scénario suivant : la soute St1 contient initialement 68m^3 tandis que la soute St2 contient 10m^3. La stabilité du navire étant en jeu, il est nécessaire de réduire l'écart de volume stocké entre les deux soutes. Le niveau dans la soute St2 étant très bas, un transfert de St1 vers St2 doit être réalisé.

Le modèle de simulation correspondant est généré en quelques secondes et comporte 1273 équations et 1273 variables. Une fois l'exécutable généré, la solution initiale du modèle est obtenue au bout d'une seconde environ. Suite à l'exécution de la simulation, l'analyse des résultats nous indique que le procédé est passé d'une configuration initiale de Stockage St1 et Stockage St2 à une configuration de Transfert St1 vers St2. Il s'agit notamment de la configuration dans laquelle l'opération de générateur est réalisée par la pompe H1.

Le système de contrôle-commande a donc permis d'amener le procédé dans un état qui permet d'atteindre les objectifs. Néanmoins, durant l'exécution de la simulation, le simulateur n'a pas été en mesure de respecter les contraintes de temps réel. Il s'avère qu'en moyenne il y a eu un dépassement des contraintes d'exécution de temps réel qui provoque un décalage entre le temps simulé (pas de temps du logiciel de simulation) et le temps réel de 81s en moyenne. En conséquence, le cycle automate n'a pas été respecté par le simulateur (décalage dans les synchronisations). En relançant la simulation mais avec un pas de temps de 0,01s (au lieu de 0,055s) l'écart dans les synchronisations a pu être réduit (49s en moyenne). Cependant, il reste encore trop important.

Nous avons alors tenté une nouvelle expérimentation en ne générant que le modèle de simulation de la partie utilisée du procédé lors du test. Le procédé considéré ne comporte alors qu'une seule pompe, et chaque soute ne dispose que d'une seule vanne de vidange. Le modèle de simulation correspondant, généré en quelques secondes, est constitué de 635 équations et 635 variables. Durant l'exécution de la simulation on observe encore un dépassement des contraintes d'exécution de temps réel. Néanmoins, cette fois-ci l'écart moyen entre le temps simulé et le temps réel est de 18s en moyenne. Bien que l'écart ait été réduit, le cycle automate n'est toujours pas respecté pour la synchronisation.

Une synthèse concernant la génération des différents modèles de simulation est représentée Table 8.1. Nous pouvons remarquer que quelle que soit la taille du modèle de simulation généré, celui-ci est toujours obtenu en moins d'une minute. Ceci montre bien que l'obtention des modèles de simulation est facilitée. Cependant, bien que nos propositions facilitent la mise en oeuvre de tests, le constat est là : nous sommes rapidement confrontés à des limites liées au matériel et/ou logiciel de simulation.

TABLE 8.1 – Synthèse des modèles de simulation générés

nbre d'éléments du procédé	*taille du modèle*	*temps d'obtention*	*respect temps réel*
31	1273 équations	$<$ 1min	non
21	635 équations	$<$ 1min	non
13	298 équations	$<$ 1min	oui

8.2.2 Conclusion

Dans un souci didactique, nous avons en premier lieu appliqué une première série de tests sur un cas d'école. Ceci nous a permis de clairement illustrer la mise en oeuvre de notre méthodologie pour interpréter les résultats issus de la simulation. En effet, lors de la

détection d'une incohérence durant le test, il réside toujours un doute : est-ce dû au modèle de simulation ? est-ce dû au système de contrôle-commande ? (programme de commande ou IHM de supervision ?). Bien évidement, plus le modèle de simulation considéré est de petite taille, plus il est facile de lever l'ambiguïté. Notre modélisation des propriétés apporte à cet égard un support : elle permet d'appréhender le système comme un tout, tout en prenant en compte différents niveaux de détails. Par exemple, dans le cas de l'exécution du premier test (concernant les alarmes liées au remplissage de la soute), si le volume initial des soutes n'avait pas été prédéfini, l'incohérence des valeurs n'aurait pas forcément été décelée dès le départ. En effet, l'objet de la vérification concernant le déclenchement des alarmes, pour pouvoir déceler cette incohérence de volume contenue dans les soutes, il faut lors de l'analyse des résultats de la simulation confronter l'état du procédé simulé au comportement attendu et à l'état du programme de commande. C'est ce que permet notre démarche.

De plus, afin de faciliter la localisation des erreurs dans le système de conduite généré, notre démarche de vérification peut être appliquée, en premier lieu, pour vérifier le programme de contrôle-commande. Une fois la cohérence de celui-ci assurée, ces mêmes tests peuvent alors être appliqués au système de contrôle-commande (incluant l'IHM de supervision). Ceci permet alors de pouvoir s'intéresser à la cohérence sémantique entre le programme de commande et l'IHM de supervision. D'autre part, l'obtention du modèle de simulation de procédé de petite taille nécessitant moins d'une minute, il est envisageable pour des systèmes de taille réelle de commencer par effectuer des tests sur des portions de celui-ci, avant d'effectuer des tests sur le système complet. L'utilisation d'une approche ascendante de vérification offre l'avantage de pouvoir valider au fur et à mesure les différentes parties fonctionnelles du système, facilitant ainsi le diagnostic lors de l'interprétation des résultats de la simulation du procédé complet.

Nous avons, par la suite, cherché à appliquer notre approche de vérification à des procédés de taille plus conséquente. Le premier constat est que notre démarche permet de générer des modèles de simulation pour des systèmes de taille industrielle et ce en un temps raisonnable. Il est toujours de quelques secondes que le système d'équations comporte 298, 635 ou 1273 équations. En revanche, nous constatons que la taille du modèle de simulation généré a une influence sur la capacité du simulateur à résoudre le système d'équations dans le temps imparti. Ceci est à l'origine des problèmes de synchronisation avec le cycle de l'automate. Ainsi, très vite nous arrivons aux limites de notre plateforme de tests.

Au travers de ce chapitre, nous avons ainsi apporté une preuve d'usage de nos contributions. Nous avons apporté un support pour la mise en oeuvre de tests au plus tôt : en permettant de disposer très rapidement du modèle de simulation et en facilitant l'interprétation des résultats de la simulation. Néanmoins pour tester le caractère adaptable du système de conduite d'un procédé, il est nécessaire de disposer d'une plateforme de tests avec une capacité de calcul adéquate, et potentiellement un logiciel adapté à la simulation temps-réel. Des travaux sur l'optimisation des modèles de simulation pourraient également être envisagés.

Conclusions et perspectives

Face à la conception de systèmes toujours plus complexes, dans un contexte industriel de réduction des coûts et des délais, être en mesure de tester la partie de pilotage du système est un point crucial. D'autant plus lorsqu'il s'agit de systèmes sociotechniques (ouverts) où les performances du système reposent sur l'optimisation conjointe des composantes humaines et techniques. Tel est le cas des systèmes de conduite de procédés.

Néanmoins, la mise en oeuvre de boucles de tests en parallèle de l'avancement dans la conception est en contradiction avec les contraintes de réduction des coûts et des délais de conception. A cet égard, nous avons cherché à faciliter l'intégration de techniques de vérification par simulation afin de tester un système de conduite, dès le début de sa conception. En particulier nous nous sommes intéressés aux procédés de type gestion de fluide.

Rappel des contributions

L'expression des propriétés est déterminante dans la mise en oeuvre de vérifications, d'autant plus lorsqu'il s'agit de vérification fonctionnelle/comportementale d'un système de conduite de procédé. Pour tenir compte du caractère adaptable du système et de son évolution dans un environnement dynamique, notre premier apport est conceptuel : nous proposons un **formalisme de modélisation multi-niveau des propriétés dans le contexte**. Celui-ci est basé sur des concepts issus de l'ingénierie des systèmes sociotechniques et de ceux des systèmes manufacturiers reconfigurables. L'avantage de cette modélisation est qu'elle permet de représenter le contexte intentionnel relié aux moyens dont dispose le système pour atteindre les objectifs. Le comportement attendu peut ainsi être directement dérivé, par rapport au scénario de test. De même, les résultats de simulation, à des instants donnés, peuvent être projetés sur cette modélisation, facilitant ainsi la comparaison avec le comportement attendu.

A partir de cette proposition de modélisation des propriétés multi-niveaux, nous avons défini une méthodologie de vérification qui **facilite l'interprétation des résultats de la simulation**. Bien que limitant les ambiguïtés lors de l'analyse des résultats issus de la simulation, la modélisation proposée ne permet pas de représenter les potentielles priorités entre les fonctions du procédé, ni celles qui ne doivent pas être implémentées simultanément. En revanche, elle permet de vérifier la cohérence sémantique du système de contrôle-commande si les tests sont effectués en deux temps.

- Une première exécution du test où seul le programme de commande est considéré : ce qui permet de s'assurer du bon comportement pour le cas traité.
- Une seconde exécution où l'on considère le système de contrôle-commande dans son ensemble : ainsi si une erreur apparaît, elle se situe au niveau de l'IHM de supervision.

De façon plus générale, pour faciliter le diagnostic des erreurs, notre démarche peut être appliquée à différents niveaux de granularité : en ne considérant que le programme de commande, puis en considérant le programme de commande et l'IHM de supervision. L'effort requis pour ces vérifications est minimisé, car seul le comportement obtenu est susceptible de changer (le

modèle de simulation, le scénario à exécuter et le comportement attendu étant identiques).

D'autre part, afin permettre l'intégration de boucles de vérifications tout au long de la conception du système de conduite, nous avons cherché à **faciliter l'obtention des modèles de simulation**, par un apport également méthodologique. Les besoins en termes de modélisation du comportement étant dépendants des vérifications à effectuer et de l'instant où elles sont effectuées dans le processus de conception, nous avons défini une stratégie de réutilisation, en nous plaçant dans un contexte de conception orientée-objet. Afin de construire le modèle de simulation du procédé à partir des modèles de simulation de ses éléments, nous avons proposé une **structuration modulaire des modèles de simulation des éléments**. Il s'agit de construire ces derniers en dissociant la modélisation du comportement plutôt discret de la partie opérative, de celle du fluide dont le comportement est à dominante continue. L'intérêt est de pouvoir réutiliser les modèles élémentaires de simulation, en ne modifiant que la partie concernée du modèle pour les nouveaux besoins, facilitant ainsi la modélisation. Par exemple, une modélisation gros grain (sans la prise en compte du dimensionnement de la tuyauterie) utilisée dans les phases de conception préliminaire, peut être affinée par la suite, lorsque les données sont disponibles.

Nous avons également cherché à faciliter la génération du modèle de simulation, en proposant **une approche de génération automatisée** de modèles de simulation en langage Modelica. La génération se fait à partir d'une représentation de l'architecture fonctionnelle du procédé (schéma P&ID) et d'une Bibliothèque d'éléments basée sur le concept de *vue*, en utilisant les techniques de l'Ingénierie Dirigée par les Modèles. De la formalisation de notre approche de génération de modèles de simulation Modelica, nous avons proposé une implémentation de ce *flot de simulation*, en parallèle du flot de conception d'Anaxagore [Bignon, 2012]. Ainsi, à partir d'un schéma représentant l'architecture fonctionnelle du procédé (schéma P&ID) et de la Bibliothèque d'éléments, est généré en moins d'une minute le modèle de simulation correspondant dans le langage Modelica.

L'enrichissement du *flot de conception* d'Anaxagore par notre *flot de simulation* nous a permis de mettre en oeuvre des vérifications à l'intérieur du flot de conception automatisé, permettant, de fait, une vérification au plus tôt. Ceci a également permis d'apporter preuve de concept et preuve d'usage de nos contributions, en mettant en évidence les apports pratiques. Le gain de temps obtenu offre la possibilité de mettre en place des vérifications en parallèle de l'avancement de la conception, et à différents niveaux de granularité. Il est ainsi possible d'effectuer des tests unitaires de la chaîne de contrôle-commande (en ne générant le modèle de simulation que de la partie du procédé concernée), puis d'effectuer des tests d'intégrations (en générant le modèle de simulation du procédé complet). Une approche ascendante de vérification peut ainsi être mise en oeuvre, offrant l'avantage de pouvoir valider au fur et à mesure les différentes parties fonctionnelles du système, et facilitant ainsi le diagnostic lors de tests d'intégration. Cela permet d'envisager une utilisation intégrée aux activités de conception : le concepteur peut tester directement sa solution, la corriger/l'améliorer, diminuant ainsi les coûts de re-conception.

Néanmoins, bien que notre approche permette de générer rapidement le modèle de simu-

lation d'un procédé complet, la mise en oeuvre des tests requiert une plateforme de tests disposant d'une capacité de calcul adéquate, de même le simulateur doit être adapté à la simulation temps-réel de systèmes de grande dimension.

Perspectives

A l'issu de ces travaux, différentes perspectives émergent, en voici un aperçu.

Une première perspective, à court terme, consiste à outiller notre démarche d'interprétation des résultats via la modélisation des propriétés, et ce afin d'être utilisable dans un contexte industriel. Il s'agit de proposer un outil de saisie de la modélisation des propriétés, puis d'utiliser les techniques de l'IDM pour obtenir la modélisation du comportement attendu, celle du comportement obtenu, afin de les comparer.

La modélisation des propriétés pourrait également être enrichie afin de prendre en compte des contraintes de sécurités et de fonctionnement. A cet égard, l'utilisation d'un autre outil de modélisation issu du Cognitive Work Analysis nous semble un bon candidat. Il s'agit du *Contextual Activity Template*, qui est déjà utilisé pour l'allocation dynamique des fonctions (entre humain et machine) [Rauffet et al., 2015] [Rauffet et al., 2013].

Une autre perspective concerne la validation des modèles de simulation. Celle-ci étant effectuée manuellement pour l'instant, nous envisageons l'utilisation de méthodes formelles [Mesli Kesraoui, 2017] pour vérifier chaque modèle de simulation contenu dans la Bibliothèque. Des vérifications formelles étant déjà utilisées sur les modèles d'entrées du flot de conception d'Anaxagore, il nous parait donc intéressant d'étendre celles-ci aux *vues de simulation* de la Bibliothèque.

Concernant la phase de qualification d'un modèle de simulation du procédé, il faut à chaque test écrire le scénario à exécuter dans le modèle et initialiser l'état du procédé en conséquence. Un intérêt est porté sur l'automatisation de cette tâche, en rajoutant au modèle de simulation (sans la communication OPC), le scénario de test. Nous envisageons ainsi une extension du flot de simulation afin d'éviter de paramétrer manuellement le modèle généré à chaque expérimentation.

Il serait également intéressant, pour faciliter les tests, de s'intéresser à la spécification de scénarios, que ce soit par rapport à la validation du modèle de simulation ou bien par rapport au test du système de contrôle-commande.

Enfin, face aux limites matérielles rencontrées pour tester des cas industriels, il pourrait être judicieux de chercher à optimiser les modèles de simulation, et/où envisager de se placer dans le cadre de la simulation distribuée comme dans les travaux de [Hadj-Amor, 2008].

D'autres perspectives, à beaucoup plus long terme, concernent l'application de notre approche de génération dans des phases plus avancées de conception. En disposant d'une représentation plus fine du comportement du procédé, il sera notamment possible de réaliser des analyses de performances. Ceci pourra également être l'occasion de s'intéresser à l'optimisation conjointe de la dimension humaine et de la dimension technique.

Bibliographie

[Adam, 2013] Adam, M. (2013). *Génération automatique d'un observateur réalisé par simulation d'un système de production.* PhD thesis, École Centrale de Nantes.

[Adam et al., 2012] Adam, M., Cardin, O., Berruet, P., and Castagna, P. (2012). Data processing from manufacturing systems to decision support systems : propositions of alternative design approaches. *IFAC Proceedings Volumes*, 45(6) :1129–1134.

[ANSI/ISA-5.1, 1992] ANSI/ISA-5.1 (1992). 5.1 instrumentation symbols and identification.

[Arlat et al., 2006] Arlat, J., Crouzet, Y., Deswarte, Y., Fabre, J.-C., Laprie, J.-C., and Powell, D. (2006). Tolérance aux fautes. *Encyclopédie de l'informatique et des systèmes d'information. Vuibert, Paris, France*, 92.

[Arroyo et al., 2016] Arroyo, E., Hoernicke, M., Rodríguez, P., and Fay, A. (2016). Automatic derivation of qualitative plant simulation models from legacy piping and instrumentation diagrams. *Computers & Chemical Engineering*, 92 :112–132.

[Auinger et al., 1999] Auinger, F., Vorderwinkler, M., and Buchtela, G. (1999). Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop". In *Proceedings of the 31st conference on Winter simulation : Simulation—a bridge to the future-Volume 1*, pages 798–805. ACM.

[Balci et al., 2011] Balci, O., Arthur, J. D., and Ormsby, W. F. (2011). Achieving reusability and composability with a simulation conceptual model. *Journal of Simulation*, 5(3) :157–165.

[Barth and Fay, 2013] Barth, M. and Fay, A. (2013). Automated generation of simulation models for control code tests. *Control Engineering Practice*, 21(2) :218–230.

[Bartholet et al., 2004] Bartholet, R. G., Brogan, D. C., Reynolds Jr, P. F., and Carnahan, J. C. (2004). In search of the philosopher's stone : Simulation composability versus component-based software design. Technical report, DTIC Document.

[Baxter and Sommerville, 2011] Baxter, G. and Sommerville, I. (2011). Socio-technical systems : From design methods to systems engineering. *Interacting with computers*, 23(1) :4–17.

[Béguin and Cerf, 2004] Béguin, P. and Cerf, M. (2004). Formes et enjeux de l'analyse de l'activité pour la conception des systèmes de travail. *Activités*, 1(1-1).

[Bell and O'Keefe, 1994] Bell, P. C. and O'Keefe, R. M. (1994). Visual interactive simulation : A methodological perspective. *Annals of Operations Research*, 53(1) :321–342.

[Benjamin et al., 1998] Benjamin, P., Erraguntla, M., Delen, D., and Mayer, R. (1998). Simulation modeling at multiple levels of abstraction. In *Proceedings of the 30th conference on Winter simulation*, pages 391–398. IEEE Computer Society Press.

[Berruet et al., 2005] Berruet, P., Lallican, J. L., Rossi, A., and Philippe, J. L. (2005). A component based approach for the design of fms control and supervision. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 4, pages 3005–3011. IEEE.

[Berruet et al., 1999] Berruet, P., Toguyeni, A. K. A., Elkhattabi, S., and Craye, E. (1999). Tolerance evaluation of flexible manufacturing architectures. *Journal of Intelligent Manufacturing*, 10(6) :471–484.

[Bévan, 2013] Bévan, R. (2013). *Approche composant pour la commande multi-versions des systèmes transitiques reconfigurables.* PhD thesis, Université de Bretagne Sud.

[Bézivin, 2005] Bézivin, J. (2005). On the unification power of models. *Software & Systems Modeling*, 4(2) :171–188.

[Bignon, 2012] Bignon, A. (2012). *Génération conjointe de commandes et d'interfaces de supervision pour systèmes sociotechniques reconfigurables.* PhD thesis, Université de Bretagne Sud.

[Bignon et al., 2010] Bignon, A., Berruet, P., and Rossi, A. (2010). Joint generation of controls and interfaces for sociotechnical and reconfigurable systems. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 749–755. IEEE.

[Bignon et al., 2013] Bignon, A., Rossi, A., and Berruet, P. (2013). An integrated design flow for the joint generation of control and interfaces from a business model. *Computers in industry*, 64(6) :634–649.

[Bishop, 2005] Bishop, R. H. (2005). *Mechatronics : an introduction.* CRC Press.

[Boehm, 1988] Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5) :61–72.

[Cabrera et al., 2010] Cabrera, A. A., Foeken, M., Tekin, O., Woestenenk, K., Erden, M., De Schutter, B., Van Tooren, M., Babuška, R., Van Houten, F., and Tomiyama, T. (2010). Towards automation of control software : A review of challenges in mechatronic design. *Mechatronics*, 20(8) :876–886.

[Cardin, 2007] Cardin, O. (2007). *Apport de la simulation en ligne dans l'aide à la décision pour le pilotage des systèmes de production–application à un système flexible de production.* PhD thesis, Université de Nantes.

[Čerić, 1997] Čerić, V. (1997). Visual interactive modeling and simulation as a decision support in railway transport logistic operations. *Mathematics and computers in simulation*, 44(3) :251–261.

[Cetinkaya and Verbraeck, 2011] Cetinkaya, D. and Verbraeck, A. (2011). Metamodeling and model transformations in modeling and simulation. In *Proceedings of the Winter Simulation Conference*, pages 3048–3058. Winter Simulation Conference.

[Cetinkaya et al., 2010] Cetinkaya, D., Verbraeck, A., and Seck, M. D. (2010). Applying a model driven approach to component based modeling and simulation. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 546–553. IEEE.

[Chapurlat, 2007] Chapurlat, V. (2007). *Vérification et validation de modèles de systèmes complexes : application à la Modélisation d'Entreprise.* Habilitation à diriger des recherche, Université Montpellier II-Sciences et Techniques du Languedoc.

[Christophe et al., 2010] Christophe, F., Bernard, A., and Coatanéa, É. (2010). RFBS : A model for knowledge representation of conceptual design. *CIRP Annals-Manufacturing Technology*, 59(1) :155–158.

[Combacau, 1998] Combacau, M. (1998). Contribution à la surveillance hiérarchisée des systèmes complexes. *Habilitation à Diriger les Recherches, université Paul Sabatier*.

[Combacau et al., 2000] Combacau, M., Berruet, P., Charbonnaud, K. A., and Zamai, E. (2000). Supervision and monitoring of production systems. *Proceedings of MCPL'2000*.

[Combemale, 2008] Combemale, B. (2008). *Approche de métamodélisation pour la simulation et la vérification de modèle–Application à l'ingénierie des procédés*. thèse, Institut National Polytechnique de Toulouse-INPT.

[Dahl et al., 2016] Dahl, M., Bengtsson, K., Bergagård, P., Fabian, M., and Falkman, P. (2016). Integrated virtual preparation and commissioning : supporting formal methods during automation systems development. *IFAC-PapersOnLine*, 49(12) :1939–1944.

[Daniel Allegro et al., 2014] Daniel Allegro, B., Le Put, A., and Tucoulou, J.-C. (2014). Introduction au penser système.

[Darses and Reuzeau, 2004] Darses, F. and Reuzeau, F. (2004). 24. participation des utilisateurs à la conception des systèmes et dispositifs de travail. In *Ergonomie*, pages 405–420. Presses Universitaires de France.

[De Lamotte et al., 2006] De Lamotte, F. F., Berruet, P., and Philippe, J.-L. (2006). Evaluation of reconfigurable manufacturing systems configurations using tolerance criteria. In *IEEE Industrial Electronics, IECON 2006-32nd Annual Conference on*, pages 3715–3720. IEEE.

[Dobre et al., 2010] Dobre, D., Morel, G., and Gouyon, D. (2010). Improving human-system digital interaction for industrial system control : some systems engineering issues. In *10th IFAC Workshop on Intelligent Manufacturing Systems, IMS'10*, page CDROM.

[Drath et al., 2008] Drath, R., Weber, P., and Mauser, N. (2008). An evolutionary approach for the industrial introduction of virtual commissioning. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 5–8. IEEE.

[El Haouzi et al., 2007] El Haouzi, H., Pannequin, R., and Thomas, A. (2007). Génération automatique de plateformes de simulation pour des systèmes organisés en flux tirés. In *7e Congrès International de Génie Industriel*.

[Elmqvist et al., 2001] Elmqvist, H., Mattsson, S. E., and Otter, M. (2001). Object-oriented and hybrid modeling in modelica. *Journal Européen des systèmes automatisés*, 35(4) :395–404.

[Estefan et al., 2007] Estefan, J. A. et al. (2007). Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group*, 25(8).

[Faisandier, 2011] Faisandier, A. (2011). Ingénierie des systèmes complexes. *Techniques de l'Ingénieur*.

[Fiorèse and Meinadier, 2012] Fiorèse, S. and Meinadier, J.-P. (2012). Découvrir et comprendre l'ingénierie système. *CEPADUES Editions*.

[Fishwick, 1995] Fishwick, P. A. (1995). Simulation model design. In *Proceeding of the 1995 Winter Simulation Conference*, pages 209–211.

[Fishwick, 1998] Fishwick, P. A. (1998). A taxonomy for simulation modeling based on programming language principles. *IIE transactions*, 30(9) :811–820.

[Folcher and Rabardel, 2004] Folcher, V. and Rabardel, P. (2004). 15. hommes, artefacts, activités : perspective instrumentale. In *Ergonomie*, pages 251–268. Presses Universitaires de France.

[Fontanili et al., 2008] Fontanili, F., Castagna, P., Yannou, B., and Thierry, C. (2008). Les outils de simulation. *La simulation pour la gestion des chaınes logistiques. Paris : Lavoisier*, pages 339–381.

[Foures, 2015] Foures, D. (2015). *Validation of simulation models* . Theses, Universite Toulouse III Paul Sabatier.

[Foures et al., 2012] Foures, D., Albert, V., Pascal, J.-C., and Nketsa, A. (2012). Automation of sysml activity diagram simulation with model-driven engineering approach. In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium*, page 11. Society for Computer Simulation International.

[Frantz, 1995] Frantz, F. K. (1995). A taxonomy of model abstraction techniques. In *Proceedings of the 27th conference on Winter simulation*, pages 1413–1420. IEEE Computer Society.

[Fratczak et al., 2015] Fratczak, M., Nowak, P., Klopot, T., Czeczot, J., Bysko, S., and Opilski, B. (2015). Virtual commissioning for the control of the continuous industrial processes—case study. In *Methods and Models in Automation and Robotics (MMAR), 2015 20th International Conference on*, pages 1032–1037. IEEE.

[Gero, 1990] Gero, J. S. (1990). Design prototypes : a knowledge representation schema for design. *AI magazine*, 11(4) :26.

[Gero and Kannengiesser, 2004] Gero, J. S. and Kannengiesser, U. (2004). The situated function–behaviour–structure framework. *Design studies*, 25(4) :373–391.

[Goubali, 2017] Goubali, O. (2017). *Apport des techniques de programmation par démonstation dans une démarche de génération automatique d'applicatifs de contrôle-commande*. PhD thesis, Ecole Nationale Supérieure de Mécanique et d'Aérotechnique.

[Goubali et al., 2014] Goubali, O., Bignon, A., Berruet, P., Girard, P., and Guittet, L. (2014). Anaxagore, an example of model-driven engineering for industrial supervision. In *Proceedings of the 2014 Ergonomie et Informatique Avancée Conference-Design, Ergonomie et IHM : quelle articulation pour la co-conception de l'interaction*, pages 58–65. ACM.

[Hadj-Amor, 2008] Hadj-Amor, H. J. (2008). *Contribution au prototypage virtuel de systèmes mécatroniques basé sur une architecture distribuée HLA. Expérimentation sous les environnements OpenModelica-OpenMASK*. PhD thesis, Toulon.

[Hardebolle, 2008] Hardebolle, C. (2008). *Composition de modèles pour la modélisation multi-paradigme du comportement des systèmes*. PhD thesis, Université Paris Sud-Paris XI.

[Hardebolle and Boulanger, 2009] Hardebolle, C. and Boulanger, F. (2009). Exploring multi-paradigm modeling techniques. *Simulation*, 85(11-12) :688–708.

[Hirtz et al., 2002] Hirtz, J., Stone, R. B., McAdams, D. A., Szykman, S., and Wood, K. L. (2002). A functional basis for engineering design : Reconciling and evolving previous efforts.

[Hoernicke et al., 2015] Hoernicke, M., Fay, A., and Barth, M. (2015). Virtual plants for brown-field projects. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8. IEEE.

[Hoffmann et al., 2010] Hoffmann, P., Schumann, R., Maksoud, T. M., and Premier, G. C. (2010). Virtual commissioning of manufacturing systems a review and new approaches for simplification. In *ECMS*, pages 175–181.

[Hutchinson et al., 2011] Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011). Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 471–480. ACM.

[INCOSE, 2011] INCOSE (2011). *Systems engineering handbook : A guide for system life cycle processes and activities*. International Council of Systems Engineering.

[Isermann and Balle, 1997] Isermann, R. and Balle, P. (1997). Trends in the application of model-based fault detection and diagnosis of technical processes. *Control engineering practice*, 5(5) :709–719.

[Johnstone et al., 2007] Johnstone, M., Creighton, D., and Nahavandi, S. (2007). Enabling industrial scale simulation/emulation models. In *Proceedings of the 39th conference on Winter simulation : 40 years ! The best is yet to come*, pages 1028–1034. IEEE Press.

[Kanso et al., 2009] Kanso, M., Berruet, P., and Philippe, J. L. (2009). Multicriteria decision making approach for reconfigurable manufacturing systems. In *European Annual Conference on Human Decision Making and Manual Control*, pages 37–44.

[Kim et al., 2013] Kim, Y. S., Shin, K. Y., Lee, J. H., Lee, S. S., Kim, K. S., Kang, K. C., and Yang, J. S. (2013). Application of virtual commissioning technology in a steel making industry. In *Control, Automation and Systems (ICCAS), 2013 13th International Conference on*, pages 1718–1720. IEEE.

[Kolski et al., 1993] Kolski, C., De Keyser, V., and Millot, P. (1993). Ingénierie des interfaces homme-machine(conception et évaluation). *Traité des nouvelles technologies*.

[Kostenko, 2017] Kostenko, A. (2017). *Evaluation multidimensionnellle et dynamique de la maitrise de la situation par l'opérateur*. PhD thesis, Université de Bretagne Sud.

[Lallican, 2007] Lallican, J. (2007). *Proposition d'une approche composant pour la conception de la commande des systèmes transitiques*. PhD thesis, Université de Bretagne Sud.

[Lallican et al., 2007] Lallican, J. L., Berruet, P., Rossi, A., and Philippe, J. L. (2007). A component-based approach for conveying systems control design. In *4th International Conference on Informatics in Control, Automation and Robotics ICINCO 2007*, pages 329–336.

[Laprie, 2008] Laprie, J.-c. (2008). From dependability to resilience. In *In 38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*. Citeseer.

[Le Moigne, 2015] Le Moigne, J. (2015). Chronique de "la théorie du système général théorie de la modélisation". *http ://www.intelligence-complexite.org/fileadmin/docs/1505jlmdg.pdf*.

[Lee and Park, 2014] Lee, C. G. and Park, S. C. (2014). Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering*, 1(3) :213–222.

[Li et al., 2013] Li, X., Lei, Y., Wang, W., Wang, W., and Zhu, Y. (2013). A dsm-based multi-paradigm simulation modeling approach for complex systems. In *Proceedings of the 2013 Winter Simulation Conference : Simulation : Making Decisions in a Complex World*, pages 1179–1190. IEEE Press.

[Maropoulos and Ceglarek, 2010] Maropoulos, P. G. and Ceglarek, D. (2010). Design verification and validation in product lifecycle. *CIRP Annals-Manufacturing Technology*, 59(2) :740–759.

[McGregor, 2002] McGregor, I. (2002). The relationship between simulation and emulation. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 2, pages 1683–1688. IEEE.

[Meinadier, 2009] Meinadier, J. (2009). Découvrir et comprendre l'ingénierie système-version expérimentale-version 3. *Association Française d'Ingénierie Système (AFIS)*.

[Mesli et al., 2016] Mesli, S., Bignon, A., Kesraoui, D., Toguyeni, A., Oquendo, F., and Berruet, P. (2016). Vérification formelle de chaines de contrôle-commande d'éléments de conception standardisés. In *Proceedings of the 11th International Conference on Modeling, Optimization & Simulation (MOSIM 2016)*.

[Mesli Kesraoui, 2017] Mesli Kesraoui, S. (2017). *Intégration des techniques de vérification formelle dans une approche de conception des systèmes de contrôle-commande : Application aux architectures SCADA*. PhD thesis, Université de Bretagne Sud.

[Mesli-Kesraoui et al., 2016a] Mesli-Kesraoui, S., Kesraoui, D., Oquendo, F., Bignon, A., Toguyeni, A., and Berruet, P. (2016a). Formal verification of software-intensive systems architectures described with piping and instrumentation diagrams. In *Software Architecture : 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings 10*, pages 210–226. Springer.

[Mesli-Kesraoui et al., 2016b] Mesli-Kesraoui, S., Toguyeni, A., Bignon, A., Oquendo, F., Kesraoui, D., and Berruet, P. (2016b). Formal and joint verification of control programs and supervision interfaces for socio-technical systems components. *IFAC-PapersOnLine*, 49(19) :426–431.

[Miller et al., 2004] Miller, J. A., Baramidze, G. T., Sheth, A. P., and Fishwick, P. A. (2004). Investigating ontologies for simulation modeling. In *Simulation Symposium, 2004. Proceedings. 37th Annual*, pages 55–63. IEEE.

[Mooz and Forsberg, 2006] Mooz, H. and Forsberg, K. (2006). 10.2. 1 the dual vee–illuminating the management of complexity. In *INCOSE International Symposium*, volume 16, pages 1368–1381. Wiley Online Library.

[Morin, 2015] Morin, E. (2015). *Introduction à la pensée complexe*. Le Seuil.

[Mosterman and Vangheluwe, 2004] Mosterman, P. J. and Vangheluwe, H. (2004). Computer automated multi-paradigm modeling : An introduction. *Simulation*, 80(9) :433–450.

[Muzy and Hill, 2011] Muzy, A. and Hill, D. R. (2011). What is new with the activity world view in modeling and simulation ? : using activity as a unifying guide for modeling and simulation. In *Proceedings of the Winter Simulation Conference*, pages 2887–2899. Winter Simulation Conference.

[Naikar et al., 2005] Naikar, N., Hopcroft, R., and Moylan, A. (2005). Work domain analysis : Theoretical concepts and methodology. Technical report, DTIC Document.

[Nance, 1993] Nance, R. E. (1993). A history of discrete event simulation programming languages. In *The Second ACM SIGPLAN Conference on History of Programming Languages*, HOPL-II, pages 149–175, New York, NY, USA. ACM.

[Niel and Craye, 2002] Niel, E. and Craye, E. (2002). *Maîtrise des risques et sûreté de fonctionnement des systèmes de production*. Lavoisier.

[OMG SysML, 2015] OMG SysML (2015). OMG Systems Modeling Language (OMG SysML) specification v. 1.4.

[Oppelt and Urbas, 2014] Oppelt, M. and Urbas, L. (2014). Integrated virtual commissioning an essential activity in the automation engineering process : From virtual commissioning to simulation supported engineering. In *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE*, pages 2564–2570. IEEE.

[Oppelt et al., 2014] Oppelt, M., Wolf, G., Drumm, O., Lutz, B., Stöß, M., and Urbas, L. (2014). Automatic model generation for virtual commissioning based on plant engineering data. In *Proceedings of the 19th World Congress of the International Federation of Automatic Control*.

[Ören, 2011a] Ören, T. (2011a). A critical review of definitions and about 400 types of modeling and simulation. *SCS M&S Magazine*, 2(3) :142–151.

[Ören, 2011b] Ören, T. (2011b). The many facets of simulation through a collection of about 100 definitions. *SCS M&S Magazine*, 2(2) :82–92.

[Ören, 2014] Ören, T. (2014). Coupling concepts for simulation : A systematic and comprehensive view and advantages with declarative models. *International Journal of Modeling, Simulation, and Scientific Computing*, 5(02) :1430001.

[Ören and Waite, 2010] Ören, T. and Waite, B. (2010). Modeling and simulation body of knowledge index : an invitation for the final phases of its preparation. *SCS M&S Magazine*, 1(4).

[Ören, 1991] Ören, T. I. (1991). Dynamic templates and semantic rules for simulation advisors and certifiers. In *Knowledge-based simulation*, pages 53–76. Springer.

[Ören, 2007] Ören, T. I. (2007). The importance of a comprehensive and integrative view of modeling and simulation. In *Proceedings of the 2007 Summer Computer Simulation Conference*, pages 996–1006. Society for Computer Simulation International.

[Page and Opper, 1999] Page, E. and Opper, J. (1999). Observations on the complexity of composable simulation. In *Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 553–560. IEEE.

[Page et al., 2000] Page, E. H., Buss, A., Fishwick, P. A., Healy, K. J., Nance, R. E., and Paul, R. J. (2000). Web-based simulation : revolution or evolution ? *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 10(1) :3–17.

[Pannequin, 2007] Pannequin, R. (2007). *Proposition of a benchmarking environment of product-driven control architectures*. Theses, Université Henri Poincaré - Nancy I.

[Pfeiffer et al., 2003] Pfeiffer, A., Kádár, B., and Monostori, L. (2003). Evaluating and improving production control systems by using emulation. *Applied Simulation and Modelling*.

[Pidd, 2002] Pidd, M. (2002). Reusing simulation components : simulation software and model reuse : a polemic. In *Proceedings of the 34th conference on Winter simulation : exploring new frontiers*, pages 772–775. Winter Simulation Conference.

[Pidd, 2004] Pidd, M. (2004). Simulation worldviews : so what ? In *Proceedings of the 36th conference on Winter simulation*, pages 288–292. Winter Simulation Conference.

[Prat et al., 2017] Prat, S., Cavron, J., Kesraoui, D., Philippe, R., Berruet, P., and Bignon, A. (2017). An automated generation approach of simulation models for checking control/-monitoring system. *IFAC-PapersOnLine*, 50(1) :6202–6207.

[Prat et al., 2016] Prat, S., Rauffet, P., Berruet, P., and Bignon, A. (2016). A multi-level requirements modeling for sociotechnical system simulation-based checking. In *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*, pages 001850–001855. IEEE.

[Prat et al., 2015] Prat, S., Rauffet, P., Bignon, A., and Berruet, P. (2015). Vers l'intégration d'une approche de génération automatique de modéle de simulation dans un flot de conception de contrôle-commande. In *JDJN MACS*.

[Pritsker et al., 1991] Pritsker, A. A. B., Henriksen, J. O., Fishwick, P. A., and Clark, G. M. (1991). Principles of modeling (panel). In *Proceedings of the 23rd conference on Winter simulation*, pages 1199–1208. IEEE Computer Society.

[Puntel-Schmidt and Fay, 2015] Puntel-Schmidt, P. and Fay, A. (2015). Levels of detail and appropriate model types for virtual commissioning in manufacturing engineering. *IFAC-PapersOnLine*, 48(1) :922–927.

[Puntel Schmidt and Fay, 2015] Puntel Schmidt, P. and Fay, A. (2015). Transformation of continuous simulation models of automated manufacturing systems into discrete event models on different levels of detail. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–5. IEEE.

[Rabardel, 1995] Rabardel, P. (1995). *Les hommes et les technologies ; approche cognitive des instruments contemporains*. Armand Colin.

[Rabardel, 2005] Rabardel, P. (2005). 13. instrument, activité et développement du pouvoir d'agir. In *Entre connaissance et organisation : l'activité collective*, pages 251–265. La Découverte.

[Rasmussen et al., 1994] Rasmussen, J., Pejtersen, A. M., and Goodstein, L. P. (1994). Cognitive systems engineering.

[Rauffet et al., 2015] Rauffet, P., Chauvin, C., Morel, G., and Berruet, P. (2015). Designing sociotechnical systems : a cwa-based method for dynamic function allocation. In *Proceedings of the European Conference on Cognitive Ergonomics 2015*, page 21. ACM.

[Rauffet et al., 2013] Rauffet, P., Morel, G., Berruet, P., and Chauvin, C. (2013). Approche pluridisciplinaire pour la conception des systèmes sociotechniques. *Journal national de la recherche en IUT*, 4 :1–12.

[Ray, 2003] Ray, C. (2003). *ATLAS, une plate-forme pour la modélisation et la simulation de systemes désagrégés*. PhD thesis, Université Rennes 1.

[Rechard, 2015] Rechard, J. (2015). *Introduction de critères ergonomiques dans une démarche de génération automatique d'interfaces de supervision*. PhD thesis, Université de Bretagne-Sud.

[Rechard et al., 2015] Rechard, J., Bignon, A., Berruet, P., and Morineau, T. (2015). Verification and validation of a work domain analysis with turing machine task analysis. *Applied ergonomics*, 47 :265–273.

[Reese and Wyatt, 1987] Reese, R. and Wyatt, D. L. (1987). Software reuse and simulation. In *Proceedings of the 19th conference on Winter simulation*, pages 185–192. ACM.

[Revel et al., 2004] Revel, L., Habchi, G., and Maire, J. (2004). Analyse du processus d'élaboration d'un projet de simulation. In *MOSIM'04*, volume 1, pages 293–301.

[Ribot, 2009] Ribot, P. (2009). *Vers l'intégration diagnostic/pronostic pour la maintenance des systèmes complexes*. PhD thesis, Université Paul Sabatier-Toulouse III.

[Riera et al., 2009] Riera, B., Vigario, B., Chemla, J.-P., Correia, L., and Gellot, F. (2009). 10 ans de maquettes virtuelles pour l'enseignement des automatismes : de winsim en 1998 à its plc professional edition en 2008. *J3eA*, 8 :1004.

[Robinson, 2004] Robinson, S. (2004). Simulation : the practice of model development and use. *John Wiley & Sons*.

[Robinson, 2008a] Robinson, S. (2008a). Conceptual modelling for simulation part i : definition and requirements. *Journal of the operational research society*, 59(3) :278–290.

[Robinson, 2008b] Robinson, S. (2008b). Conceptual modelling for simulation part ii : a framework for conceptual modelling. *Journal of the Operational Research Society*, 59(3) :291–304.

[Robinson, 2012] Robinson, S. (2012). Choosing what to model-conceptual modeling for simulation. In *Proceedings of the Winter Simulation Conference*, page 168. Winter Simulation Conference.

[Robinson, 2015] Robinson, S. (2015). A tutorial on conceptual modeling for simulation. In *Proceedings of the 2015 Winter Simulation Conference*, pages 1820–1834. IEEE Press.

[Robinson et al., 2004] Robinson, S., Nance, R. E., Paul, R. J., Pidd, M., and Taylor, S. J. (2004). Simulation model reuse : definitions, benefits and obstacles. *Simulation modelling practice and theory*, 12(7) :479–494.

[Roca et al., 2015] Roca, R., Pace, D., Robinson, S., Tolk, A., and Yilmaz, L. (2015). Paradigms for conceptual modeling. In *Proceedings of the 48th Annual Simulation Symposium*, pages 202–209. Society for Computer Simulation International.

[Rochet, 2007] Rochet, S. (2007). *Formalisation des processus de l'Ingénierie Système : Proposition d'une méthode d'adaptation des processus génériques à différents contextes d'application.* PhD thesis, Institut National des Sciences Appliquées de Toulouse.

[Ruault et al., 2009] Ruault, J., Colas, C., Sarron, J., and Luzeaux, D. (2009). Ingénierie système et résilience des systèmes sociotechniques. In *5e Conférence Annuelle d'Ingénierie Système AFIS 2009.*

[Sargent, 2001] Sargent, R. (2001). Some approaches and paradigms for verifying and validating simulation models. In *Simulation Conference, 2001. Proceedings of the Winter*, volume 1, pages 106–114. IEEE.

[Sargent, 2011] Sargent, R. G. (2011). Verification and validation of simulation models. In *Proceedings of the 2011 Winter Simulation Conference (WSC).*

[Schiess, 2001] Schiess, C. (2001). Emulation : debug it in the lab-not on the floor. In *Simulation Conference, 2001. Proceedings of the Winter*, volume 2, pages 1463–1465. IEEE.

[Schludermann et al., 2000] Schludermann, H., Kirchmair, T., and Vorderwinkler, M. (2000). Soft-commissioning : hardware-in-the-loop-based verification of controller software. In *Proceedings of the 32nd conference on Winter simulation*, pages 893–899. Society for Computer Simulation International.

[Süß et al., 2015] Süß, S., Strahilov, A., and Diedrich, C. (2015). Behaviour simulation for virtual commissioning using co-simulation. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–8. IEEE.

[Toguyéni et al., 2003] Toguyéni, A. K. A., Berruet, P., and Craye, E. (2003). Models and algorithms for failure diagnosis and recovery in fmss. *International Journal of Flexible Manufacturing Systems*, 15(1) :57–85.

[Vangheluwe et al., 2002] Vangheluwe, H., De Lara, J., and Mosterman, P. J. (2002). An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems)*, pages 9–20.

[Varenne, 2010] Varenne, F. (2010). Framework for m&s with agents in regard to agent simulations in social sciences. *Activity-Based Modeling and Simulation*, pages 53–84.

[Varenne, 2014] Varenne, F. (2014). Chapitre 1. épistémologie des modèles et des simulations : tour d'horizon et tendances. *Modélisations, simulations, systèmes complexes*, pages 13–46.

[Vepsäläinen et al., 2008] Vepsäläinen, T., Hästbacka, D., and Kuikka, S. (2008). Tool support for the uml automation profile-for domain-specific software development in manufacturing. In *Software Engineering Advances, 2008. ICSEA'08. The Third International Conference on*, pages 43–50. IEEE.

[Vepsäläinen and Kuikka, 2014] Vepsäläinen, T. and Kuikka, S. (2014). Model-driven development of automation and control applications : modeling and simulation of control sequences. *Advances in Software Engineering*, 2014 :3.

[Versteegt and Verbraeck, 2002] Versteegt, C. and Verbraeck, A. (2002). Real-time control : the extended use of simulation in evaluating real-time control systems of agvs and automated material handling systems. In *Proceedings of the 34th conference on Winter simulation : exploring new frontiers*, pages 1659–1666. Winter Simulation Conference.

[Vicente, 1999] Vicente, K. J. (1999). *Cognitive work analysis : Toward safe, productive, and healthy computer-based work*. CRC Press.

[Villemeur, 1988] Villemeur, A. (1988). *Sûreté de fonctionnement des systèmes industriels*. Eyrolles.

[Wagner et al., 2016] Wagner, G., Seck, M., and McKenzie, F. (2016). Process modeling for simulation : Observations and open issues. In *Winter Simulation Conference (WSC), 2016*, pages 1072–1083. IEEE.

[Wagner et al., 1996] Wagner, P., Freitas, C., and Wagner, F. (1996). A new paradigm for visual interactive modelling and simulation. In *Proceedings of the 1996 8th European Simulation Symposium*, pages 142–145.

[Zeigler et al., 2000] Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation : integrating discrete event and continuous complex dynamic systems*. Academic press.

[Zeigler et al., 1995] Zeigler, B. P., Song, H. S., Kim, T. G., and Praehofer, H. (1995). Devs framework for modelling, simulation, analysis, and design of hybrid systems. In *Hybrid Systems II*, pages 529–551. Springer.

Annexes

A

Squelette du modèle ModeleSimuSyst

```
<?xml version="1.0" encoding="UTF-8"?>
<modelica:Modelica xmlns:modelica="http://www.segula.fr/simulation/modelica">
  <package domain="PlateForme">
    <package system="Auxiliaires">
      <model subSystem="EdS">
        <!--- Les instances des modèles de simulation des éléments-->
        <component name="V2VM" dirSimu="V2VM/Verif/D_VSimu/" id="V2VM01" fami
        </component>
         ...
        <!--- Les instances des modèles de simulation d'un équipotentiel-->
        <component name="Equipotentiel" dirSimu="" id="equipotentiel8" family
            <variable type="Integer" name="N" value="3"></variable>
        </component>
         ...
        <!--- Les variables globales instanciees -->
        <variable type="Integer" name="ffdcC_V2VM01" comment="Defaut capteur
         ...
        <!--- Les connexions -->
        <equation>
            <!--- Les connexions des variables globales -->
            <calcul value="ffdcC_V2VM01 = V2VM01.ffdcC"></calcul>
             ...
            <!--- Les connexions des equipotentiels -->
            <connect>
                <element cptName="equipotentiel8" portNum="[1]" typePort="sta
                <element cptName="H1" portNum="1" typePort="stand"></element>
            </connect>
             ...
            <!--- Les connexions de l'instrumentation -->
            <connect>
                <element cptName="St1" portNum="3" typePort="mesure"></elemen
                <element cptName="LT0001" portNum="1" typePort="mesure"></ele
            </connect>
             ...
        </equation>
      </model>
```

FIGURE A.1 – Extrait du squelette du modèle de simulation ModeleSimuSyst.xml

B

Connexion des variables OPC

Prenons l'exemple d'une variable booléenne en mode lecture, c'est à dire une interface dont `type`="BOOL" et `kind`="InPhy" (par exemple l'interface CmdO de V2VM01). Le modèle à instancier est, par conséquent, le modèle *Read* qui est contenu dans le *package* associée aux booléens, à savoir *SamplerBoolean*. L'instanciation d'une telle variable OPC, se fait via une balise `<component>` dont l'attribut `name` prend alors la valeur "SamplerBoolean.Read". La valeur de l'attribut `id` est quant-à-elle donnée en suivant la même convention que pour nommer les variables globales. Ainsi, l'instance de la variable OPC associée à l'interface CmdO de V2VM01 est la balise `<component>` où `id`="CmdO_V2VM01" (Figure 7.28). Contrairement à l'interface CmdO, l'interface FdcO de V2VM01 est de nature *OutPhy*, la valeur de l'attribut `name` de l'instance de sa variable OPC (`id`="FdcO_V2VM01") est donc *SamplerBoolean.Write*.

Afin de créer un lien entre une variable OPC et l'interface de simulation, le modèle de la variable OPC dispose d'une interface dédiée. Il s'agit de l'interface y (représentant le résultat de la lecture sur le serveur) pour le mode lecture, et de l'interface u (représentant l'entrée à écrire sur le serveur) pour le mode écriture.

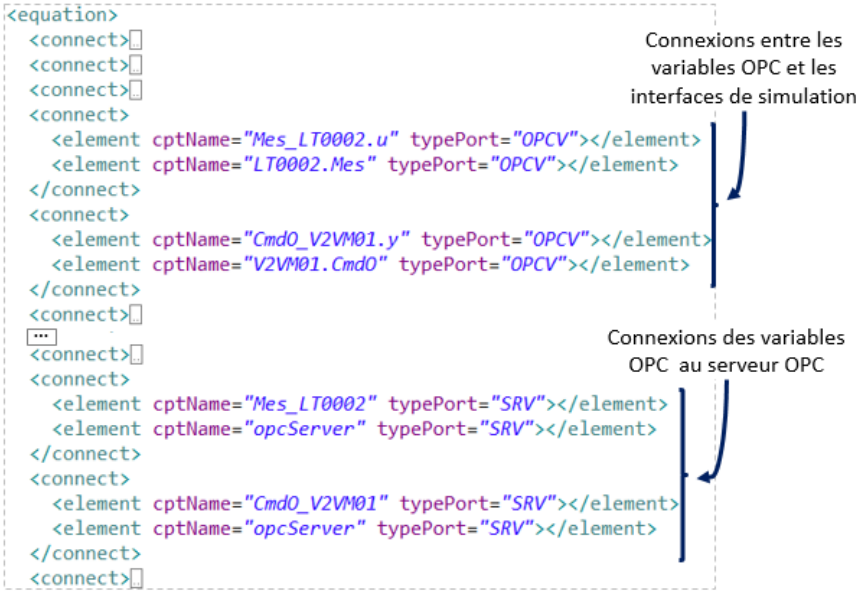


FIGURE B.1 – Extrait du modèle VarOPCInst : connexions des variables OPC

Prenons, par exemple, l'instance CmdO_V2VM01 du modèle Read. Son lien avec l'interface CmdO de l'instance V2VM01 (*V2VM01.CmdO*) se fait en y connectant l'interface *CmdO_ V2VM01.y*. Cette connexion est ainsi définie par les balises `<connect> </connect>` (Figure B.1) qui contiennent une balise `<element>` où `cptName`="CmdO_V2V01.y", et une balise `<element>` où `cptName`="V2VM01.CmdO". Pour l'instance Mes_LT0002 (instance du modèle Write), il s'agit de connecter l'interface *Mes_ LT0002.u* à l'interface Mes de l'instance LT0002 (*LT0002.Mes*).

Pour achever d'instancier la communication OPC, les connexions entre les instances des variables OPC et l'instance du serveur OPC doivent être créées. La définition d'une connexion se fait via une balise `<element>` dont l'attribut `cptName` correspond au nom de l'instance du modèle de la variable OPC, et une seconde où cet attribut correspond au nom de l'instance du serveur. La connexion entre l'instance CmdO_V2VM01 et le serveur OPC (Figure B.1), est ainsi définie par une balise `<element>` dont `cptName`="CmdO_V2VM01" et une seconde où `cptName`="opcServer".

C

Exemple de traduction en code modelica

Instance du serveur OPC

La balise `<component>` associée à l'instance du modèle du serveur OPC (c'est à dire celle dont `name`="OPCServer") contient une balise `<variable>`. Il s'agit dans ce cas de définir en argument de l'instance le nom du serveur OPC (attribut `name`="serverName") (Figure C.1).

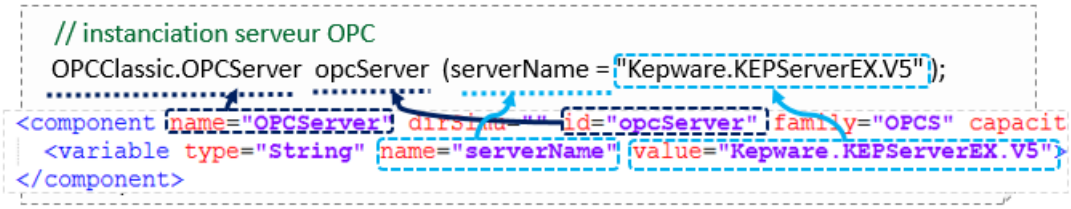


FIGURE C.1 – Traduction en code Modelica : instance du serveur OPC

Connexion d'une variable OPC à l'interface de simulation de l'élément

Considérons maintenant une connexion entre une variable OPC et l'interface de simulation de l'élément. Prenons par exemple la mesure du niveau effectuée par le capteur LT00020. Il s'agit de connecter la variable `Mes` de LT0002 à la variable `u` de Mes_LT0002 (Figure C.2). Ainsi, `variable1` correspond à `Mes_LT0002.u` et `variable2` à `LT0002.Mes`. Le code associé est donc « `connect(Mes_LT0002.u,LT0002.Mes);` ».

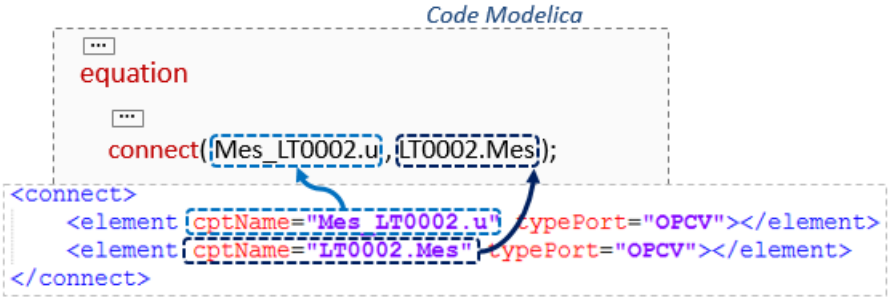


FIGURE C.2 – Traduction en code Modelica : instance du serveur OPC

D

Validation des alarmes : état initial de la commande

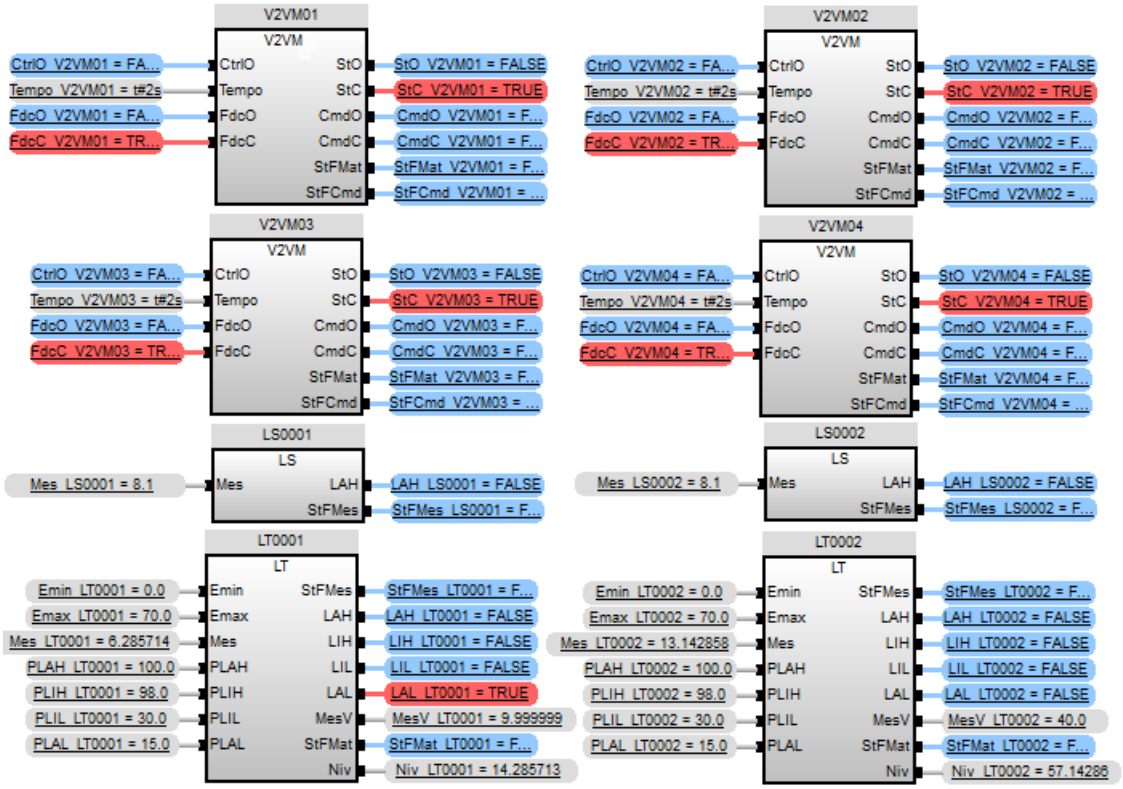


FIGURE D.1 – Etat initial du programme de commande

Intégration de techniques de vérification par simulation dans un processus de conception automatisée de contrôle-commande

Aujourd'hui, la conception ne porte plus sur de simple objets, mais sur des systèmes complexes, sociotechniques et ouverts. Les systèmes de conduite de procédés font partie de ce type de systèmes, où les performances du système reposent sur l'optimisation conjointe des composantes humaines et techniques. Afin de limiter la détection d'erreur tardive, il devient alors important de pouvoir effectuer des tests tout au long de la conception, sans augmenter les coûts et les délais de conception.
L'objectif de nos travaux est de faciliter l'intégration de techniques de vérification par simulation, dès le début de la conception, pour des systèmes de conduite de procédés de type gestion de fluide. Pour tenir compte du caractère adaptable du système et de son évolution dans un environnement dynamique, une première contribution porte sur la démarche de vérification, basée sur la formalisation et la contextualisation des propriétés à vérifier. Puis, afin de faciliter l'obtention des modèles de simulation du procédé nécessaires à la mise en oeuvre des vérifications tout au long de la conception, nous proposons une approche de génération automatisée des modèles de simulation du procédé dans le langage Modelica (modélisation multi-domaine), à partir d'un schéma P&ID (représentation de l'architecture fonctionnelle du procédé) et d'une bibliothèque d'éléments (contenant les modèles de simulation des éléments). L'implémentation de cette approche dans le cadre du flot de conception automatisé de contrôle-commande d'Anaxagore permet d'apporter une preuve de concept et une preuve d'usage de nos propositions.

Mots clés: Vérifications fonctionnelles, Système sociotechnique, Système de conduite, Simulation Software-int-the-loop, Ingénierie Dirigée par les Modèles, Modelica.

Integration of simulation-based checking into an automated design approach of control-monitoring system

Nowadays, engineers have to design open, complex and sociotechnical systems. The process control systems belong to this class of systems, in which the system performance relies on the joint optimisation of technical componants and human componants. To avoid the late discovery of design errors, it is necessary to perform tests throughout the design without adding design costs and delays.
The aim of this work is therefore to facilitate the integration of checking by simulation, from early design stage, for process control systems such as fluid management systems. Regarding the adaptable feature of the system and its evolution in a dynamic environment, a first contribution focusses on the verification approach, by modelling the requirements within the context. Then, to facilitate the obtaining of the process simulation models required for checking throughout the design, we propose an automatic generation approach of simulation models in Modelica language (multi-domain modelling), from a P&ID model (modelling of the functional architecture of the process) and a library of elements (containing the simulation models of elements). To provide a proof of concept and a proof of use of our proposals, this approach has been implemented into Anaxagore, an automated design flow for monitoring and control.

Keywords: Functional verifications, Sociotechnical system, Process control system, Software-in-the-loop simulation, Model Driven Engineering, Modelica.

n°d'ordre : 476
Université de Bretagne Sud
Centre de recherche Christiaan Huygens - rue de Saint Maudé - 56321 Lorient Cedex
Tèl : + 33(0)2 97 87 45 60