Table des matières

i. Table des figures

ii. Table des tableaux

Chapitre 1
Introduction

Sommaire

1.1 Contexte du travail

La miniaturisation des composants électroniques a permis de concevoir des puces de plus en plus petites, plus performantes, consommant moins d'énergie et offrant un nombre de fonctionnalités toujours croissant [1]. Les puces électroniques se retrouvent généralement au centre de systèmes complexes et sont chargées de gérer les données provenant de différents capteurs et modules. Par exemple, le microcontrôleur STM32 F2 peut se retrouver au centre de la montre connectée *Pebble*. Bien que ce circuit ne mesure que 16mm², il centralise les données des différents capteurs de mouvement (LIS3DH de STMicroelectronics) et gère la connexion avec un téléphone par *Bluetooth* tout en offrant des performances et une autonomie de batterie importante.

Ces systèmes centraux regroupant à la fois du logiciel et du matériel sur une même puce sont appelés Systèmes sur Puce (ou SoC pour *System on Chip*). Un SoC est comparable à un ordinateur classique, à la différence que les éléments d'un ordinateur sont réparties sur plusieurs cartes (carte son, carte graphique, carte mère, RAM …) tandis que le SoC contient tous ces éléments sur une seule et même puce. Nous pouvons aisément comprendre la complexité de réalisation de ce type de puce où tous les éléments sont développés et intégrés en une seule fois.

Le développement d'un système sur puce nécessite l'intervention de plusieurs centaines d'acteurs répartis autour du monde. Outre les raisons économiques de cette distribution, cela permet de spécialiser chaque équipe sur un aspect particulier du développement (marketing,

développement matériel, développement logiciel, documentation…) et ainsi paralléliser les développements. Ces différents intervenants vont s'intéresser à différents aspects ou points de vue particuliers du système à développer, nécessitant une expertise sur différents corps de métier et la gestion d'outils et de langages spécifiques permettant d'exprimer leurs idées. En effet, le développement d'un SoC peut nécessiter l'utilisation de plus de cinquante formats de documents différents.

1.2 Problématique abordée

Le développement d'un système sur puce complexe nécessite de considérer à la fois les aspects fonctionnels et extra-fonctionnels tel que la performance temporelle, la consommation énergétique, la dissipation de chaleur, l'encombrement, la sureté ou encore la sécurité. Les activités de conception font donc intervenir des expertises variées, s'appuyant sur des modèles et des outils spécifiques. En conséquence, nous assistons à une multiplication de documents dans des formats variés, qui saisissent de manière partiellement redondante la même information.

Dans le contexte du développement de SoC complexe, il est important de noter que les différents documents utilisés sont liés entre eux. Les documents créés par les intervenants à différentes étapes du flot dupliquent de l'information venant de documents préexistants. Par exemple, un développeur créant un document de code source décrivant le comportement d'un registre d'un SoC se réfère forcément à une spécification créée en amont. La modification du code source ou de la spécification est donc susceptible d'impacter l'autre document, nécessitant par conséquent l'allocation de temps supplémentaire afin de synchroniser les informations des deux documents (ou maintenir la cohérence).

De plus, le domaine de l'électronique et des semi-conducteurs est régi par un contexte économique et concurrentiel très fort : le système développé doit être prêt le plus rapidement, être le moins couteux et afficher une qualité suffisante afin de s'assurer un bon temps de mise sur le marché et un bon prix unitaire.

La présence d'incohérences dans le corpus documentaire impliqué dans le développement d'un SoC est susceptible d'impacter négativement la fonctionnalité du produit, sa qualité et le temps de mise sur le marché. En effet, si l'implémentation d'un SoC n'est pas cohérente avec la spécification ou la documentation, la complexité de débogage en sera majorée, et nécessitera de nombreuses itérations afin de synchroniser les documents, entrainant une dépense de temps supplémentaire. De plus, si le SoC livré au client n'est pas cohérent avec la documentation, l'image de la qualité du fournisseur sera négativement impactée, causant une baisse des ventes du produit. Il est donc nécessaire de maintenir la cohérence du corpus documentaire tout au long du cycle de développement d'un SoC.

Cependant, aujourd'hui, maintenir la cohérence est un réel challenge. En effet, les liens reliant les documents sont généralement implicites ou uniquement présent dans l'esprit des

différents acteurs qui ont une vague idée de qui utilise leurs documents. De plus, les formats de document et surtout les différentes structures de ces documents empêchent l'utilisation systématique de programme permettant de traduire les documents dans d'autres formats et la distribution physique des équipes apportent une difficulté supplémentaire pour maintenir la cohérence.

1.3 Contributions

L'approche présentée dans ce manuscrit est issue d'une thèse CIFRE réalisée en collaboration entre STMicroelectronics et le CEA LIST. Elle vise à améliorer la qualité des systèmes sur puce développés en supprimant les incohérences pouvant exister entre les documents décrivant ce système. Nous présentons quatre contributions principales :

- **Une analyse des modèles de cohérence appliqués au flot de conception des systèmes sur puce**. Cette analyse se base sur une analogie entre les systèmes distribués et le flot de conception et nous permet de remarquer qu'aujourd'hui le modèle de cohérence appliqué dans le flot de conception est implicite et généralement faible (*eventual consistency*) garantissant uniquement que les documents seront cohérents au bout d'un certain temps (inconnu).
- **Un moyen d'expliciter la description d'architecture d'un système initialement implicites.** La description d'architecture d'un système est composée d'une part du corpus documentaire décrivant le système et d'autre part des différentes actions effectués par les acteurs du flot sur ces documents. Nous avons tout d'abord analysé la définition des transformations de modèles issue du domaine de l'Ingénierie Dirigée par les Modèles (ou IDM). De cette analyse, nous proposons une extrapolation de la définition afin de considérer chaque modification de document comme une transformation. Cette extrapolation nous permet d'utiliser des travaux sur la caractérisation des transformations pour définir les liens de cohérence inter-documentaire. La formalisation des liens s'appuie sur des extensions du standard de description d'architecture ISO/IEC/IEEE 42010 [2] afin de faire apparaitre clairement les propriétés définissant les liens de cohérence.
- **Une méthodologie d'exploitation de la description d'architecture pour garantir un modèle de cohérence explicite.** Cette méthodologie énonce, selon nous, les meilleures pratiques d'utilisation de la description d'architecture et des extensions que nous proposons, tant au niveau de sa capture que de la taille des fragments liables, dans le but de propager les modifications intervenant entre les documents. De plus, l'exploitation des liens nous permet de faire émerger le processus réellement suivi par les différents acteurs du flot de conception et ce uniquement en observant les pratiques de travail.
- **Le développement d'un prototype mettant en œuvre l'approche.** Ce prototype permet à la fois d'illustrer, sur un cas d'étude concret extrait des pratiques de

STMicroelectronics, et d'évaluer l'efficacité de l'approche au travers de deux variables : le nombre d'incohérences détectées et corrigées dans un corpus documentaire et le temps moyen nécessaire à la correction.

Les travaux présentés dans ce manuscrit ont fait l'objet d'une communication à la *Design Automation Conference* en 2016 et d'un article accepté à la conférence *Modelsward* en 2017 [3].

1.4 Organisation du document

Ce document est structuré en trois parties, subdivisé en huit chapitres, chacun d'eux dédié à décrire un aspect du travail réalisé. La Figure 1-1 donne un aperçu de la structure de ce document.

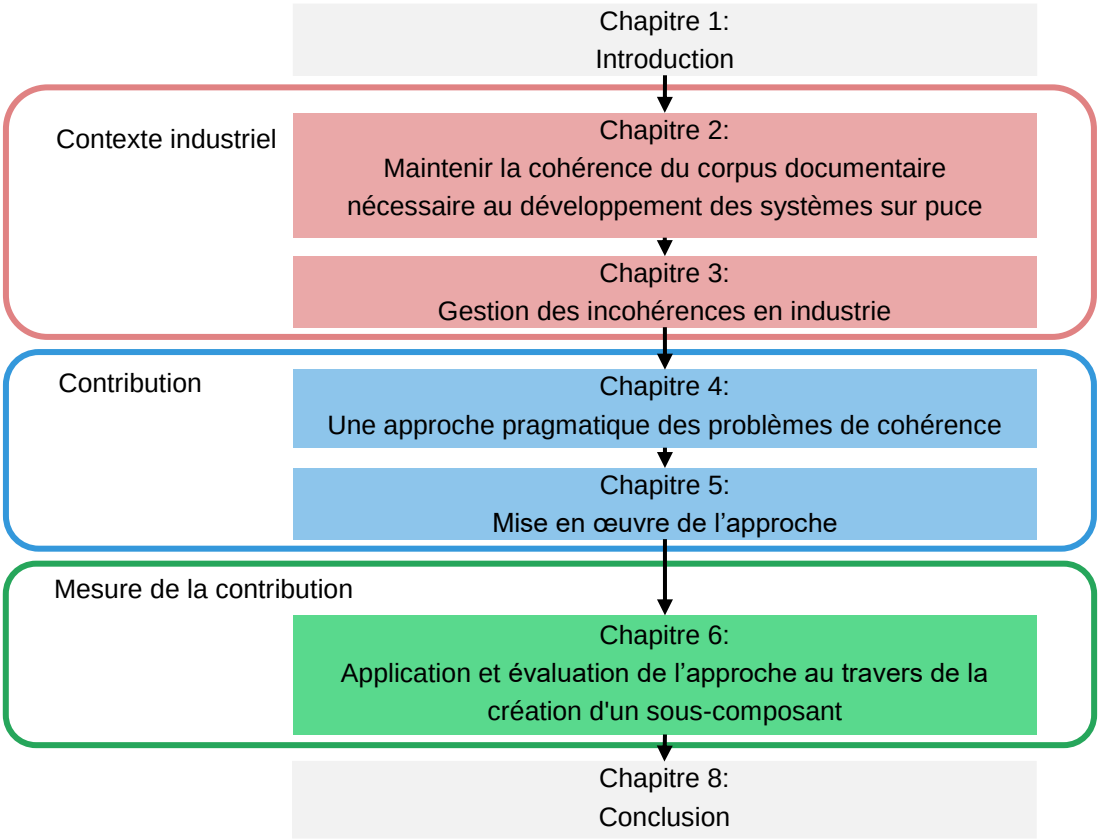


FIGURE 1-1 STRUCTURE DU DOCUMENT

Après l'introduction, la première partie vise à analyser le contexte industriel de notre travail de thèse. Cette partie est composée de deux chapitres :

- Dans le **Chapitre 2**, nous présentons le flot de conception des systèmes sur puces et analysons quelles sont les spécificités pouvant devenir des points bloquants pour maintenir la cohérence du corpus documentaire.
- Dans le **Chapitre 3**, nous nous focalisons sur les différents outils et méthodes mis en œuvre dans STMicroelectronics pour gérer les spécificités du flot de

conception des systèmes sur puce en indiquant leurs limitations vis-à-vis de la maintenance de cohérence.

La deuxième partie de ce mémoire constitue le cœur de cette thèse. C'est en effet dans cette partie que nous explicitons l'approche proposée. Cette partie est divisée en deux chapitres :

- Dans le **Chapitre 4**, nous proposons une méthodologie, basée sur le concept de description d'architecture, aidant les développeurs à maintenir systématiquement la cohérence entre les documents. Cette approche est définie en deux temps. Premièrement un modèle est défini afin de décrire formellement les liens entre les documents. Le modèle est indépendant des formats de documents, du cycle de développement et des méthodes de travail de l'entreprise. Deuxièmement, ces liens sont analysés et utilisés afin d'aider les différents intervenant à maintenir la cohérence des documents en les informant lorsqu'un document est modifié.
- Dans le **Chapitre 5,** nous explicitons la conception et l'articulation des différentes parties du prototype mettant en œuvre l'approche.

Enfin la troisième partie de ce manuscrit propose une évaluation cherchant à valider l'intérêt de l'approche au travers d'un cas d'étude extrait des méthodes de travail de STMicroelectronics. Cette partie est composée d'un chapitre :

- Dans le **Chapitre 6** nous présentons en détail le cas d'étude. Ce cas d'étude présente la création d'un *Intellectual Property block* (ou *IP block*) et nous permet de proposer une évaluation de l'approche. L'évaluation a pour but de quantifier l'efficacité de l'approche, au travers de 2 variables : le taux de détection et de correction d'incohérence et le temps moyen dépensé pour corriger une incohérence.

Le dernier chapitre conclut ce mémoire en rappelant les points essentiels de notre contribution, discutant des points à compléter et, plus généralement, en proposant des ouvertures pour des travaux futurs.

Chapitre 2
Maintenir la cohérence du corpus documentaire nécessaire au développement des systèmes sur puce

Sommaire

Aujourd'hui, les systèmes sur puces développés par STMicroelectronics sont composés de plusieurs processeurs, d'une centaine de blocs de propriété intellectuelle, de dizaines de millions de lignes de codes et peuvent mobiliser plusieurs centaines d'employés. Le temps de mise sur le marché (*Time To Market* ou TTM) des systèmes sur puce est une contrainte de développement très importante afin de s'assurer une bonne part de marché et un bon prix unitaire du produit.

Maintenir la cohérence du corpus documentaire est primordial afin de conserver un bon TTM. Les incohérences sont sources de bogues qu'il faudra corriger, impactant donc le temps de développement et le cout du projet. Les documents, créés par les différents acteurs impliqués dans le développement d'un système sur puce, ne sont pas indépendants les uns des autres. En effet, les nouveaux documents sont généralement créés en se référant à des documents pré-existants. Dans ce contexte, la modification d'un document est susceptible d'impacter la cohérence du corpus documentaire, causant une perte de temps afin de réconcilier les documents. Nous entendons par les termes "réconciliation" ou "réalignement" l'action de synchronisation nécessaire afin qu'aucun document n'en contredise un autre.

De plus, en observant les méthodes de travail de STMicroelectronics, nous avons identifié trois points qui pourront devenir bloquants dans les prochaines années, si les méthodologies de maintenance de cohérence et de propagation de changements utilisées ne sont pas changées. Ces caractéristiques semblent inhérentes au développement de systèmes complexes comme le confirme Bézivin *et al.* dans [4] :

- De nombreux formats de documents sont nécessaires afin d'exprimer et de spécifier les différents aspects d'un système sur puce mais cette hétérogénéité accroit la difficulté de maintenance de cohérence des documents.
- La distribution des tâches et des équipes est obligatoire afin de faire décroitre le temps de mise sur le marché des produits mais ce parallélisme des tâches nécessite une allocation de temps de la part des employés afin de résoudre les problèmes de réconciliation des documents.
- Les méthodes agiles et les processus itératifs sont appliqués afin de gérer plus efficacement la complexité toujours croissante des systèmes et pour s'adapter plus rapidement aux changements de spécifications. Cependant, ces pratiques basent la maintenance de cohérence sur des réunions fréquentes entre les équipes, ce qui n'est pas toujours possible dans le contexte d'une grande entreprise où les équipes sont réparties tout autour du monde. Ceci nécessite des méthodes ou des outils de propagation de changements appropriés.

Dans ce chapitre, nous débuterons par présenter le développement des systèmes sur puce par STMicroelectronics en exposant la complexité de tels systèmes. Nous décrirons ensuite le processus de développement des systèmes sur puces afin d'expliciter les spécificités de ce flot. Enfin, nous définirons une analogie entre les systèmes distribués et le flot de conception des systèmes sur puce, nous permettant d'analyser la notion de cohérence dans notre contexte.

2.1 Complexité des systèmes sur puces

La Figure 2-1, présente l'architecture, sous forme de diagramme blocs, d'une puce STiH252 (extrait de [5]) conçue par STMicroelectronics que l'on peut retrouver au cœur des décodeurs satellites Sky Q [6].

Son architecture est en de nombreux points semblables à celle que l'on peut retrouver

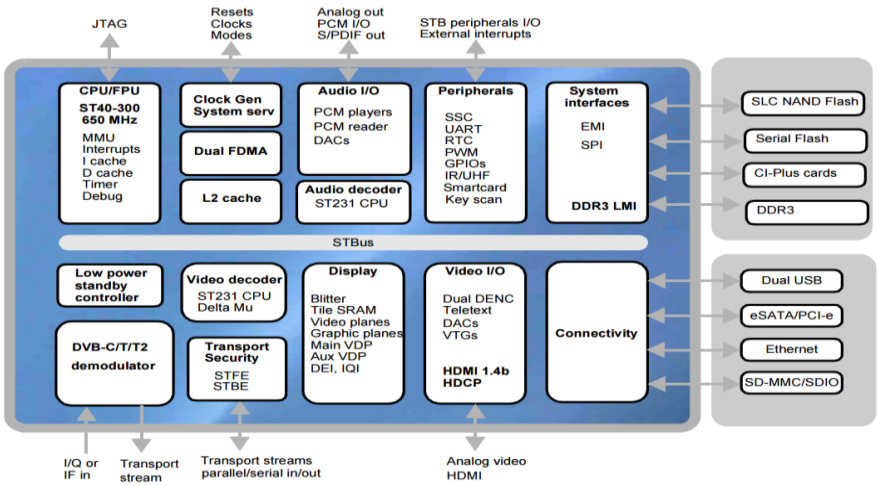


FIGURE 2-1 DIAGRAMME BLOCS DU STIH252

dans les ordinateurs classiques. Nous retrouvons ainsi un microcontrôleur (ST40-300) accompagné de ses différents niveaux de mémoire cache et contenant également de nombreux composants classiques tels que les *Timers* (pour la gestion du temps), un contrôleur d'interruption (*Interrupt*) et de nombreux contrôleurs d'entrées/sorties (comme USB, Ethernet, mémoires, flash, cartes SD, Sata, Audio, vidéo analogique, HDMI). Nous retrouvons également des blocs spécifiques à des applications particulières, comme le décodage du flux audio et vidéo (*Audio decoder* et *Video decoder*) tous deux basés sur un microcontrôleur ST231, un démodulateur de signal numérique (DVB-C/T/T2 *demodulator*) ou pour des composants de traitement d'images (*Blitter*, *Graphic planes*, *Tile SRAM*).

Afin de pouvoir inter-opérer, tous ces éléments sont interconnectés entre eux (*ST Bus*). À la différence d'un ordinateur classique, où ces éléments seraient répartis sur des puces et cartes différentes assemblées entre elles (carte mère, carte son, carte graphique, barrettes de RAM), ici tous les éléments sont rassemblés sur une seule et unique puce. De plus, il est important de noter que les éléments de la Figure 2-1 peuvent également être des sous-systèmes. Par exemple, nous observons que le décodeur vidéo ou le décodeur audio sont, eux-mêmes, composés d'un microcontrôleur ST231, donc contenant à leur tour de la mémoire, des périphériques, des *timers*, des registres d'interruptions (cf. Figure 2-2 extrait de [7]).

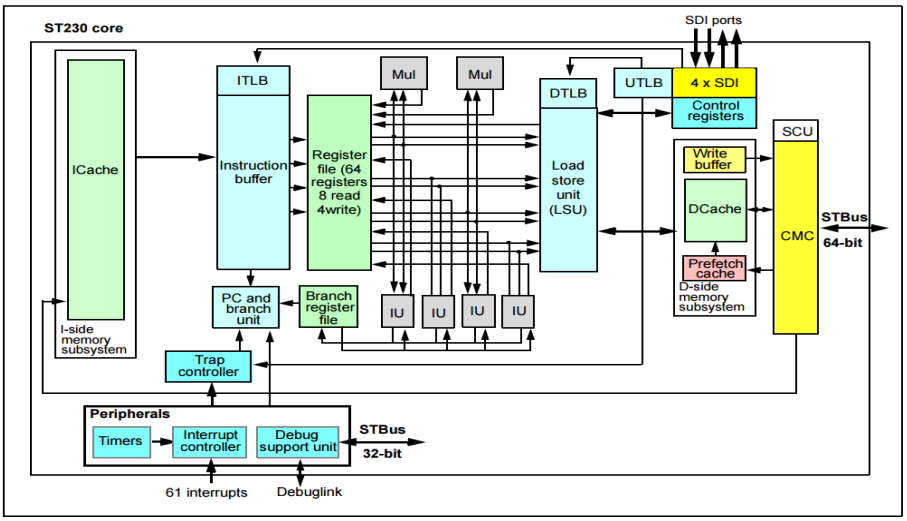


FIGURE 2-2 DIAGRAMME BLOCS DU ST231

Nous pouvons aisément comprendre la complexité de réalisation de ce type de puce. Pour un ordinateur, différents constructeurs fournissent des composants n'ayant qu'une seule fonction et dont les connexions sont clairement définies par des interfaces et des protocoles standardisés (par exemple PCI Express). En revanche, pour les systèmes sur puce les éléments sont développés et intégrés en une seule fois.

2.2 Présentation du flot de conception des systèmes sur puce

Nous nous intéressons maintenant au flot de réalisation des systèmes sur puce qui peut se résumer par la Figure 2-3. Avant de rentrer dans le détail des différentes étapes, nous pouvons déjà remarquer que ce processus se décompose en deux phases principales : la phase de spécifications et la phase de modélisation et d'implémentation du système.

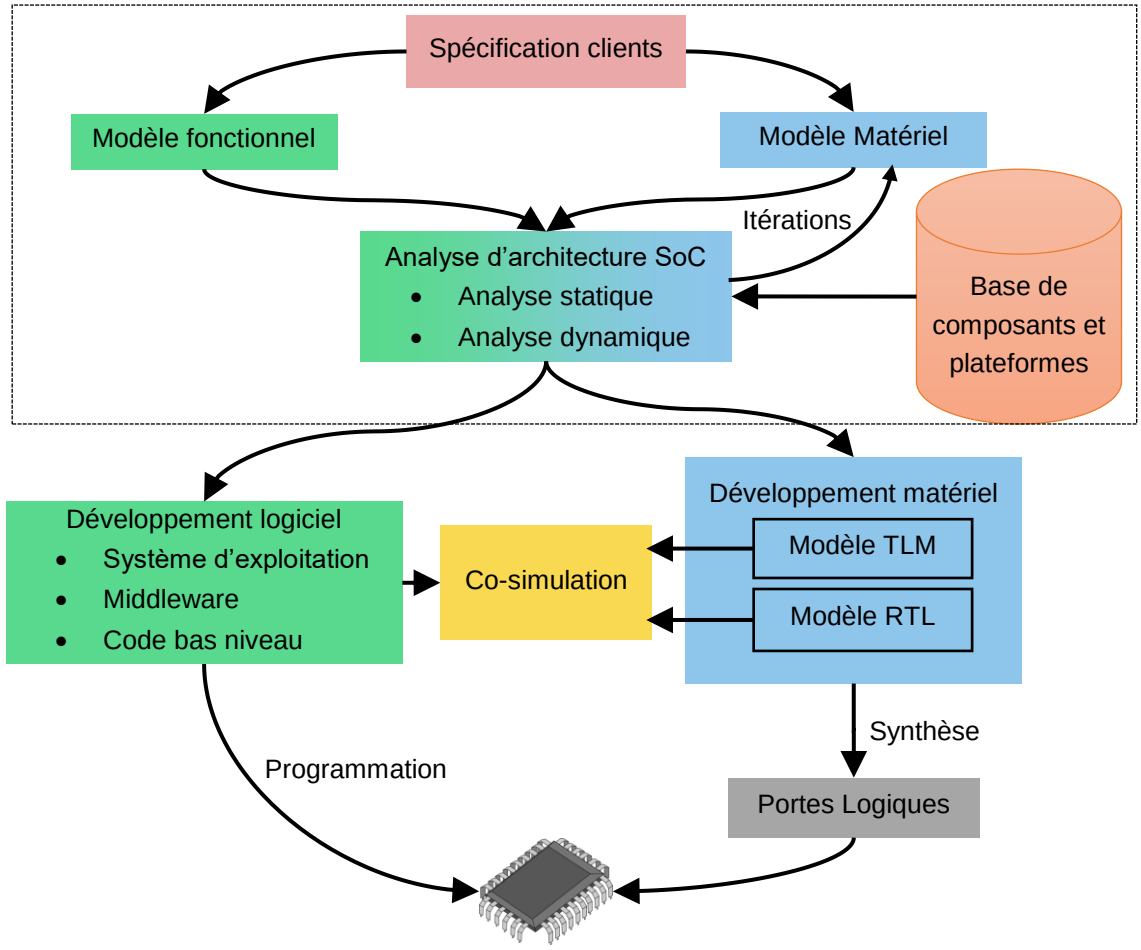


FIGURE 2-3 REPRÉSENTATION SIMPLIFIÉE DU FLOT DE CONCEPTION DES SYSTÈMES SUR PUCE

Dans la phase de spécification (allant des spécifications clients jusqu'à l'analyse d'architecture), les exigences du client, que devra remplir la puce, sont identifiées et exprimées, tout d'abord avec une granularité gros grain de manière textuelle, puis en rentrant dans le détail des fonctionnalités devant être accomplies. La phase de spécifications se termine par l'analyse d'architecture du système sur puce (*System on Chip* ou SoC). Au cours de cette étape, l'architecte détermine quelle sera la configuration matérielle, en termes de blocs de propriétés intellectuelles assemblés sur la puce (comme sur la Figure 2-2), afin de répondre au mieux aux exigences exprimées par le client. Cette étape se termine par un

partitionnement Matériel/Logiciel qui va consister à déterminer quelle proportion de la réalisation des fonctionnalités de la puce sera attribuée au matériel et au logiciel.

À la fin de cette étape, l'architecture du système est déterminée, la phase d'implémentation débute par deux branches parallèles : le développement logiciel et le développement matériel. Le développement matériel se concentre sur la partie physique de la puce et se réalise à travers différents niveaux de modélisation, dont les deux principaux que nous donnons ici : le niveau transactionnel (*Transaction Level Modeling* ou TLM) [8] et le niveau registre (*Register Transfert Level* ou RTL) [9]. Le niveau RTL est obligatoire et permet de manière très concrète de décrire le fonctionnement de l'intégralité des composants présents sur la puce. Le modèle RTL pourra être automatiquement synthétisé afin de générer l'ensemble des opérations logiques élémentaires (portes logiques) dont l'implémentation est réalisée par des transistors et des pistes gravées sur le circuit.

Il est important de noter que le développement du logiciel peut se faire en même temps que le développement du matériel grâce à la co-simulation. En effet, la partie logicielle définie lors de la phase de spécification est souvent très dépendante de la partie matérielle et surtout du fonctionnement des composants matériels. Le logiciel ne peut donc pas être développé et exécuté sur des ordinateurs classiques. Il faudrait théoriquement attendre que la puce soit entièrement réalisée avant de commencer le développement du logiciel. Cependant, ceci entrainerait un temps avant la mise sur le marché de la puce beaucoup trop long. En outre, l'exécution du logiciel fait très souvent apparaitre des défauts de spécification ou de conception du matériel. C'est pourquoi des techniques ont été développées afin de permettre au logiciel de s'exécuter sur des modélisations du matériel. En effet, le principal intérêt du modèle TLM est de pouvoir exécuter le logiciel sur un modèle haut-niveau du composant, plus rapidement et plus tôt que sur les modèles RTL.

Nous pouvons désormais nous intéresser plus en détail aux différentes étapes du processus de conception des systèmes sur puce afin de comprendre les enjeux et concepts qui entrent en jeu.

2.2.1 Spécifications clients

Nous commençons ce processus par l'expression des exigences que doit remplir le système sur puce. Cette expression est généralement faite, en concertation avec le client, de manière informelle à travers des documents textuels ou graphiques (par exemple MS Word, MS Powerpoint). Bien que le système sur puce rassemble un grand nombre de fonctionnalités et de sous-composants sur une seule et même puce, son but premier est d'être intégré sur des cartes électroniques contenant d'autres composants, tels que des capteurs, des périphériques d'affichage (par exemple écran LCD), des modules de puissance etc. Le système sur puce devra donc être capable de communiquer et d'interagir avec ces différents composants. En conséquence, les spécifications client portent à la fois sur les fonctionnalités que la puce elle-même devra remplir, mais également sur les contraintes de communication

que l'environnement, dans lequel le système sur puce est intégré, nous impose. Ces différentes contraintes peuvent porter sur la performance de calcul, le coût, le temps de développement, la consommation énergétique, l'encombrement, la mémoire, la sureté pour une utilisation sur des systèmes critiques ou la sécurité de protection des données (par exemple pour les puces de cartes bancaires). Ces contraintes doivent être prises en compte lors de la conception du système sur puce.

2.2.2 Modèle fonctionnel et matériel

Après la spécification des exigences, différents modèles exécutables sont implémentés afin de modéliser les différentes fonctionnalités attendues du système. Ces modèles sont purement fonctionnels ; ils décrivent les fonctions et algorithmes principaux que la puce devra réaliser, sans se soucier de l'exécution effective sur la puce. Ces modèles vont permettre d'identifier les différents blocs fonctionnels pouvant par la suite être utilisés pour l'étude de l'architecture de la puce. En effet, ces modèles peuvent à la fois intégrer des algorithmes spécialement développés pour répondre à un nouveau besoin, une nouvelle fonctionnalité, ou reprendre des modèles préexistants, créés dans un précédent projet ou extrait d'un standard, tels que le décodage de vidéos MP4 par exemple. Ces modèles sont généralement écrits dans des langages génériques comme le C ou le C++, mais peuvent également être définis grâce à des outils ou langages spécialisés tels que Matlab [10], SDL [11], Esterel [12] ou SysML [13].

2.2.3 Analyse d'architecture SoC

L'analyse de l'architecture SoC est un point très important et déterminant du reste du processus. C'est lors de cette étape que les architectes déterminent les différentes parties du système sur puce afin que celui-ci réponde aux exigences exprimées par le client. Le principal paramètre de cette étape est la séparation des parties effectuées par le logiciel et celles effectuées par le matériel.

Il existe plusieurs façons d'exécuter les algorithmes spécifiés lors des modélisations fonctionnelles et matérielles. L'algorithme peut, par exemple, être exécuté par un CPU. Dans ce cas, l'algorithme restera sous forme logicielle. Cette option est la plus simple à mettre en œuvre, et donc la moins couteuse à réaliser. De plus, cette option permet de retoucher l'algorithme même après la conception finale de la puce étant donné qu'il est possible de la reprogrammer pour effectuer d'éventuelles modifications ou corrections de l'algorithme.

En revanche, lorsque l'algorithme est trop complexe, les processeurs peuvent vite être rattrapés par leurs limitations ; ils risquent d'être trop peu puissants ou trop grands consommateurs d'énergie pour satisfaire les exigences requises. Dans ce cas, des composants ou blocs dédiés, optimisés pour le traitement d'un algorithme spécifique, peuvent être créés. Ces composants spécifiques sont beaucoup plus performants et proposent une consommation d'énergie optimale. Cependant, ces composants sont beaucoup plus

complexes à réaliser (donc un temps de développement beaucoup plus long) et une fois gravés dans le silicium il est impossible de les modifier ou de les corriger. Il est donc nécessaire que ceux-ci ne présentent pas de défauts ou de vices cachés.

Nous pouvons remarquer sur la Figure 2-3 que l'architecture SoC n'est pas définie *ex nihilo* pour chaque nouvelle puce. En effet, des blocs constituant le système sur puce sont généralement développés afin d'être réutilisables. L'architecture d'une nouvelle puce peut donc intégrer des composants ou des assemblages de composants préexistants. Ces composants ou assemblages de composants peuvent se retrouver légèrement modifiés ou adaptés à un cas d'utilisation. Ce mode de fonctionnement est appelé conception basée sur les plateformes ou *Platform Based Design* [14].

L'analyse de l'architecture SoC se termine sur la détermination d'un système où le logiciel a été spécifié, ainsi que le matériel qui devra l'exécuter.

2.2.4 Développement Logiciel

L'évolution des systèmes sur puce au cours des années, avec la généralisation des architectures multi-processeurs, a donné une part de plus en plus importante au logiciel. On en retrouve ainsi sur plusieurs niveaux. Tout d'abord, la plupart des processeurs possède un système d'exploitation (OS), qui doit être modifié pour assurer la compatibilité avec la configuration matérielle choisie et obtenir les meilleures performances du matériel. Ce portage des OS inclut par exemple le développement de code bas niveau spécifiques à la configuration matérielle permettant l'initialisation de l'OS sur la cible.

De plus, certains blocs peuvent contenir des composants dédiés associés à des processeurs, qui doivent exécuter un code embarqué (*Firmware*). Par exemple, les algorithmes du modèle fonctionnel, doivent être modifiés afin de les exécuter efficacement sur des ressources parfois limitées, d'où un travail d'optimisation et de nombreuses itérations. En reprenant la Figure 2-1, nous pouvons voir que le STiH252 contient au moins quatre processeurs indépendants, soit autant de *Firmwares* a développés : un ST40-300 dans le bloc CPU/FPU, un ST231 dans le bloc *Video decoder*, un second ST231 dans le bloc *Audio decoder* et un dans le bloc Dual FDMA.

2.2.5 Développement Matériel

Le développement matériel a pour but d'implémenter les composants, déterminés dans la phase d'analyse d'architecture SoC, qui constitueront physiquement le système sur puce. Comme pour le développement logiciel, tous les composants matériels ne sont pas développés *ex-nihilo* pour chaque nouveau système sur puce. Les développeurs vont donc tenter de réutiliser au maximum des composants préexistants (développés lors de précédents projets) ou d'en dériver des variantes, toujours dans l'optique de répondre aux exigences attendues de la puce.

Le travail de développement matériel consiste donc à réaliser de nouveaux composants issus de l'analyse d'architecture de la puce, au niveau RTL et dans un langage automatiquement synthétisable, tel que le VHDL ou le Verilog, en portes logiques combinatoires (portes ET, OU, multiplexeur etc.) et séquentielles (bascule D), et à la création de la plateforme finale intégrant ces composants nouvellement créés et les composants réutilisés. La synthèse permettra enfin d'obtenir une description du système qui après des tests et optimisations va pouvoir être envoyée en production pour la fabrication des masques qui permettront de graver la puce sur le silicium.

L'intégration des composants est un travail de développement important puisqu'il faut s'assurer que les différents composants sont aptes à interagir correctement entre eux. Les tests et vérifications ont un rôle majeur dans ce travail de développement matériel. En effet, il faut être certain que les circuits produits ne présentent pas de vices cachés ou de dysfonctionnements.

La validation du matériel se réalise par des simulations de modèles. Il existe plusieurs niveaux d'abstraction pour la modélisation des composants matériels. La Figure 2-4 présente une comparaison des performances des différents niveaux d'abstraction pour le décodage d'une image MPEG-4.

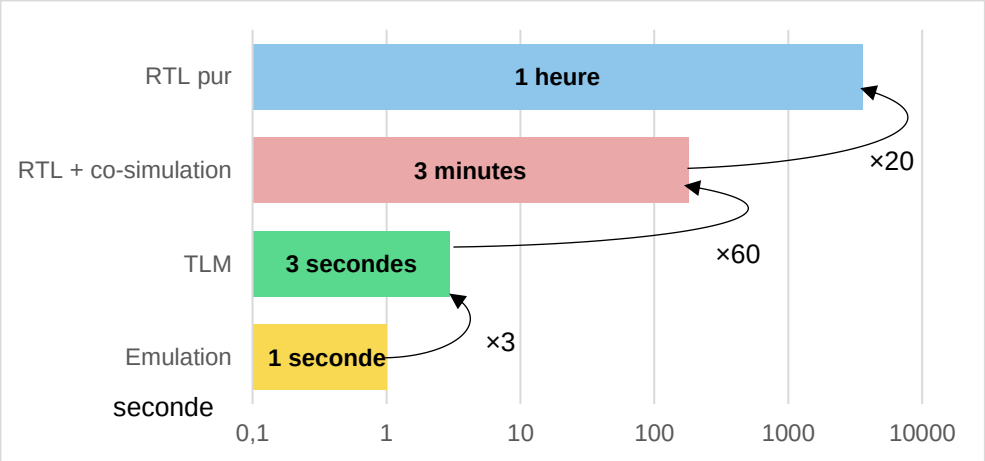


FIGURE 2-4 COMPARAISON DU TEMPS DE SIMULATION DU DÉCODAGE D'UNE IMAGE MPEG-4

Le niveau le plus complet et précis pour la simulation est le niveau RTL car il décrit l'intégralité du comportement des différents composants. Cependant, cette précision et ce niveau de détail coûtent très cher en termes de temps de simulation ; plus le système sur puce est complexe, plus le temps de simulation sera long. Par exemple, le décodage d'une image MPEG-4 se fera en environ une heure sur un modèle RTL.

Il est possible de diminuer ce temps de simulation en abstrayant le comportement de certains blocs. L'abstraction des différents blocs se fait en les modélisant avec du code C dont la simulation fournira les mêmes résultats que le modèle RTL. De cette façon, un

processeur peut être, par exemple, modélisé par un simple simulateur de jeu d'instructions, ou une mémoire par un tableau de valeurs. Cette simulation de modèle niveau RTL couplée à des modèles C est appelée co-simulation. Dans le cas du décodage d'une image MPEG-4, ce niveau d'abstraction permet de diviser par 20 le temps de simulation comparé à la simulation en RTL pur.

Il est encore possible de diminuer le temps de simulation en abstrayant la notion de temps et d'horloge du modèle. Dans ce cas, l'élément le plus fin traduisant l'évolution du système est la transaction, c'est-à-dire le transfert d'une donnée d'un composant à un autre. Ce niveau de modélisation (TLM) permet de simuler l'intégralité d'un système en un temps suffisamment court afin qu'il soit utilisé pour le développement du logiciel embarqué. Grâce à ce niveau d'abstraction le décodage d'une image MPEG-4 ne prend plus que trois secondes, soit 60 fois moins de temps qu'avec la co-simulation et 1200 fois moins qu'une simulation en RTL pur.

De plus, l'utilisation d'adaptateur permet de simuler un système majoritairement modélisé en TLM avec un composant à tester modélisé en en RTL. Ceci permet de tester rapidement ce composant dans un environnement proche de la réalité avec le logiciel embarqué devant interagir avec lui.

Pour en finir avec ce panorama des techniques de simulation, notons la possibilité d'utiliser des émulateurs matériels. Tout comme les FPGA (*Field-Programmable Gate Array*), il s'agit de composants matériels reprogrammables, c'est-à-dire que les liaisons entre cellules logiques élémentaires peuvent être modifiées électroniquement. Ils permettent donc de simuler un système comme s'il avait été synthétisé, avec un niveau de détail maximum. L'utilisation de cette technique pour décoder une image MPEG-4 ne prend plus qu'une seconde, soit trois fois moins qu'avec la simulation au niveau TLM. Cependant, les machines suffisamment puissantes pour simuler un système sur puce complexe sont malheureusement très couteuses, et n'offrent que peu de possibilités pour le débogage (contrairement au TLM). Leur principale utilisation consiste à faire tourner de larges batteries de tests de façon automatique, dans l'espoir de trouver les derniers bugs avant la création des masques.

Le développement matériel fait donc appel à au moins quatre modèles de simulation différents. Chacun de ces modèles répond à un besoin particulier (comme un temps de simulation suffisamment court ou une représentation précise de la puce) et sont développés indépendamment. Cependant, ces différents modèles référent et se basent sur la même spécification. Une modification d'un de ces modèles pourra nécessiter une modification de la spécification et donc des autres modèles, afin qu'aucun modèle n'en contredise un autre. En effet, aujourd'hui, aucun standard ne garantit que les informations contenues dans ces modèles sont alignées, quels sont les modèles impactés par une modification de la spécification et quels sont ceux mis à jour.

2.3 Conséquences des incohérences

Comme nous venons de le voir, le développement d'un système sur puce est une opération complexe, composée de nombreuses étapes et faisant appel à différents intervenants qui vont créer des documents pour décrire le système en développement. Ces différents documents encodent l'information dans des formats permettant de décrire les différents aspects du système.

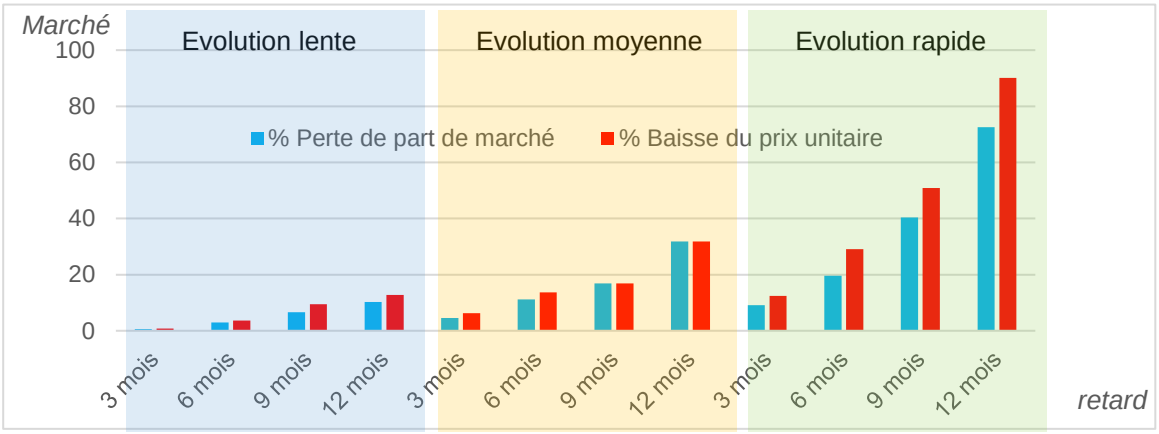
Les documents impliqués dans le développement d'un SoC ne sont pas tous indépendants les uns des autres. En effet, la production d'un nouveau document, est souvent le résultat d'un raffinement, de l'abstraction, de la composition, du référencement ou de l'extraction d'information de documents préexistants. Par exemple, en reprenant le processus présenté dans la section précédente, les choix faits lors de la création du modèle fonctionnel sont dépendants des spécifications textuelles, qui elles-mêmes sont dépendantes des exigences informelles que le client nous aura imposées. Dans ce cas, la modification d'un seul de ces documents est susceptible d'avoir une influence sur la totalité du corpus documentaire, obligeant les développeurs à allouer du temps pour propager la modification à tous les documents. En effet, modifier un document peut introduire une incohérence dans le corpus documentaire et nécessiter la propagation du changement pour maintenir la cohérence : un document modifié peut potentiellement nécessiter la modification d'un deuxième document contenant la même information afin que les deux documents restent cohérents entre eux, mais cette modification (du deuxième document) peut à son tour nécessiter la modification d'un troisième document etc.

Le cout d'un masque, permettant la gravure des transistors dans le silicium d'une puce lors de la phase finale de conception peut dépasser le million d'euros. Il est très important pour STMicroelectronics, que la puce soit entièrement opérationnelle et qu'elle réponde parfaitement aux exigences du client, tant sur le plan des fonctionnalités, que des performances ou de la consommation énergétique. Il est donc impératif, en amont de l'étape de production effective, que tous les documents (comme les spécifications, les simulations, les modèles de la puce) soient cohérents et qu'aucune incohérence ou ambiguïté ne puisse devenir une source d'erreur une fois que la puce sera produite.

En outre, la présence d'incohérence est susceptible d'impacter négativement l'image de STMicroelectronics auprès de ses clients. Par exemple, si le silicium livré n'est pas cohérent avec la documentation fournie, des bogues ou des difficultés de développement peuvent apparaitre dans l'application du client. Sa vision de la qualité des produits STMicroelectronics en pâtira. En conséquence, ce client risque de chercher un autre fournisseur plus fiable, causant une baisse des ventes pour STMicroelectronics.

De plus, le temps de mise sur le marché d'un produit électronique (tel qu'un système sur puce) est régi par le contexte économique et concurrentiel : le système doit être le moins couteux à fabriquer, prêt le plus rapidement possible afin d'avoir une part de marché et un

prix unitaire les plus élevés. En effet, l'électronique et les semi-conducteurs sont sur un marché à évolution rapide [15]. Comme nous pouvons le voir sur la Figure 2-5 (extrait de [16]), de tels marchés supportent très mal les retards, causant à la fois une perte de part de marché et une baisse du prix unitaire du produit très importantes. Par exemple, avec un retard de mise sur le marché de neuf mois, le prix unitaire baisse d'environ 50% de son prix initial et la part de marché est réduite de 40%.



Source : IBS[1]

FIGURE 2-5 IMPACT DU RETARD SUR LES REVENUS

Les incohérences dans l'ensemble des documents impliqués dans le développement d'un système sur puce peuvent être la cause d'une perte de temps à plusieurs niveaux. Par exemple, si l'implémentation d'un sous-système n'est pas alignée avec sa documentation, le débogage de sous-système en question sera beaucoup plus compliqué, entrainant donc une perte de temps. En outre, si les documents divergent au fur et à mesure du développement, le temps devant être alloué au réalignement, donc à la correction des incohérences, avant la livraison finale sera également majoré. Il est donc essentiel, pour ce genre de systèmes complexes, de maintenir la cohérence tout au long de son cycle de développement.

2.4 Spécificités du flot

Le flot de conception des systèmes sur puce nous permet de faire ressortir plusieurs points pouvant devenir bloquants quant à la maintenance de cohérence dans le corpus documentaire :

- L'hétérogénéité des formats
- La distribution des équipes
- Les développements itératifs et agiles

[1] IBS : *International Business Strategies, Inc*.

2.4.1 Hétérogénéité des formats

De nombreuses étapes sont nécessaires afin d'élaborer les différents aspects de la puce. Chaque étape fait appel à différents corps de métier, nécessitant une certaine expertise, ainsi que des outils et des langages spécifiques, qui vont permettre aux différents acteurs impliqués dans le développement du système sur puce d'enrichir la description du système. Nous donnons dans la Table 2-1 une liste non-exhaustive des langages et outils utilisés dans les différentes étapes du processus.

TABLE 2-1 RÉSUMÉ NON-EXHAUSTIF DES OUTILS ET LANGAGES UTILISÉS DANS LE FLOT DE CONCEPTION DES SYSTÈME SUR PUCE

Étape	Langages/Outils utilisés	
Spécifications clients	• MS Word • MS Excel	• MS Powerpoint • FrameMaker
Modèle fonctionnel et matériel	• C • C++ • Matlab • SDL	• Esterel • SysML • Java
Analyse d'architecture	• FrameMaker • Générateur de trafic • Simulation Cycle Accurate	• MS Word • MS Excel • MS Powerpoint
Développement logiciel	• C • C++	• Assembleur
Développement Matériel	• IP-XACT • VHDL/VHDL-AMS • Verilog/Verilog-AMS • SystemC • System Verilog • C	• C++ • MS Word • Python • Perl • Tcl
Documentation	• MS Word • PDF	• DITA

De nombreux langages et outils sont exploités. Cependant, ces différents formats de documents accroissent la complexité de propagation de changement, dégradant par là même la cohérence du corpus documentaire, ceci à cause des différents niveaux de structures. En effet, ces documents peuvent avoir des structures allant de données très structurées, présentant une syntaxe clairement définie, tel que les composants IP-XACT [17] par exemple, qui pourront facilement être interprétées informatiquement par un programme, jusqu'aux données que nous appellerons non-structurées, comme des présentations MS Powerpoint ou simplement des fichiers textes, présentant une structure implicite et non

formelle qui seront beaucoup plus difficilement utilisables par un programme. Ces différences de structures nous empêchent d'assurer automatiquement la cohérence du corpus documentaire. Ceci sous-entendrait que les données répliquées sont restées sous la même forme, ou que la syntaxe de la donnée nous offre un moyen de tracer et de générer la même information sous une autre syntaxe, ce qui n'est pas forcément notre cas. En effet, nous serions dans ce cas obligé de développer des outils permettant de traduire chaque format vers tous les autres formats, représentant alors un effort et un investissement temporel beaucoup trop important.

Par exemple, la définition d'un registre mémoire est faite sous plusieurs syntaxes différentes selon les activités l'exploitant (voir Table 2-2). Plus précisément, la taille du registre mémoire pourra être défini dans une spécification MS Word comme étant le nombre de colonne d'un tableau (une colonne par bit du registre), dans une description d'un composant IP-XACT comme étant un nombre entre deux balises « size », dans une description d'un composant VHDL comme la taille du vecteur logique, dans une définition de macro C comme étant un nombre. Il s'agit pourtant bien de la même information sémantique dans tous les cas. Dans ce manuscrit, nous utiliserons le terme d'information sémantique pour faire référence à une information pouvant se retrouver sous plusieurs syntaxes différentes mais implémentant toujours le même concept fondamental. Dans notre exemple, la taille du registre (32 bits) est une information sémantique dupliquée dans les six documents.

Dans des cas comme celui-ci, la propagation du changement et la maintenance de cohérence ne sont pas traitables automatiquement par les programmes, ou alors au prix de trop gros effort de développement ou d'un temps pour effectuer la génération trop important. En effet, malgré les avancées pour formaliser et automatiser de nombreux aspects du développement, dans notre contexte industriel de nombreuses modifications sont intrinsèquement liées à l'intellect humain, qui doit se charger d'analyser la signification d'une information, d'extraire l'information sémantique, afin de la transposer sous une autre syntaxe.

TABLE 2-2 DÉCLARATION D'UN REGISTRE DANS DIFFÉRENTS LANGAGES

Spécification TEXTE

REG_1

Configuration register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD1																															
R/W																															

Address: 0x00
Type: R/W
Reset: 0xFFFFFFFF

[31:0]	**AD1**: Used as an input register when writing new data into component

Description du registre IP-XACT

```
<spirit:register>
  <spirit:name>REG_1</spirit:name>
  <spirit:description>Configuration register 1</spirit:description>
  <spirit:addressOffset>0x00</spirit:addressOffset>
  <spirit:size>32</spirit:size>
  <spirit:access>read-write</spirit:access>
  <spirit:reset>
    <spirit:value>0xFFFFFFFF</spirit:value>
  </spirit:reset>
  <spirit:field>
    <spirit:name>AD1</spirit:name>
    <spirit:description>Used as an input register when writing new data into component</spirit:description>
    <spirit:bitOffset>0</spirit:bitOffset>
    <spirit:bitWidth>32</spirit:bitWidth>
    <spirit:access>read-write</spirit:access>
  </spirit:field>
</spirit:register>
```

Description du registre VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity testReg is

    port (
        REG_1: in std_logic_vector (31 downto 0);
    );
end testReg;
```

Définition de macro C

```
/*!
* \brief
* Register : REG_1
* Configuration register 1
*/

#define UART_DMAC_UART1_data_SIZE                    (32)
#define UART_DMAC_UART1_data_OFFSET                  (0x00)
#define UART_DMAC_UART1_data_RESET_VALUE             (0xFFFFFFFF)
#define UART_DMAC_UART1_data_BITFIELD_MASK           (0x00000000)
#define UART_DMAC_UART1_data_RWMASK                  (0xFFFFFFFF)
#define UART_DMAC_UART1_data_ROMASK                  (0x00000000)
#define UART_DMAC_UART1_data_WOMASK                  (0x00000000)
#define UART_DMAC_UART1_data_UNUSED_MASK             (0xFFFFFFFF)
```

Initialisation de la banque de registres C

```
#include "base_COMPONENT.hpp"

//------------------------------------------
// Register banks initialization
//------------------------------------------
void base_COMPONENT::init_regs_COMPONENT() {
    std::stringstream ss;
    ss << "REG_1              0x0     32b  0x00  [RW]; /* Configuration register 1 */  \n";
    regs_COMPONENT.set_register_mapping(ss);
}
```

Documentation PDF

4.4.1 Configuration register 1 (REG_1)

Address offset: 0x00

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AD1[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **Configuration register bits**
Used as an input register when writing new data into component

2.4.2 Distribution des équipes

La conception d'un système sur puce peut impliquer plusieurs centaines d'intervenants distribués tout autour du monde. En effet, STMicroelectronics est une société multinationale présente sur les cinq continents. Le travail distribué est un ensemble de personne travaillant dans le but d'un objectif commun, mais dont les frontières spatiales, temporelles, culturelles et organisationnelles diffèrent [18].

Distribuer les différentes tâches de développement d'un produit à différentes équipes permet de paralléliser le travail et donc de développer et finaliser plus rapidement un produit (*Separation of Concern* [19]). Cette pratique est spécialement utilisée dans des secteurs, comme STMicroelectronics, où les temps de mise sur le marché sont cruciaux. De plus, l'utilisation d'équipes géographiquement distribuées tend à s'accroitre pour des raisons à la fois économique, le coût de la main d'œuvre selon les régions du monde est une considération non négligeable pour une entreprise, mais également historique, comme les fusions ou les rachats d'entreprises.

D'après nos observations faites dans STMicroelectronics, la distribution du travail peut effectivement causer des conflits entre les documents, qui à leur tour peuvent entrainer des problèmes de réconciliation quand certaines informations sémantiques sont répliquées dans plusieurs documents. En effet, au cours de l'évolution du développement, les informations sont susceptibles d'être modifiées et chaque réplique de ces informations risque de diverger. Une réconciliation implique que les changements peuvent être propagés. Cependant, les outils actuels faisant de la réconciliation ne permettent pas de gérer les cas où les données ou documents à réconcilier ont été transformés dans différents formalismes. Cette situation laisse la maintenance de cohérence, qui est difficile, à une synchronisation basée sur l'humain. Cependant, l'humain montre par nature un caractère non déterministe et non fiable ; même avec les meilleures intentions du monde, un humain peut faire des erreurs. En outre, un grand nombre de propagations sont faites de manière informelle : appel téléphonique, email, meeting ou discussion autour d'un café.

Enfin, si la propagation d'une modification n'est pas faite immédiatement, les incohérences ne pourront pas impacter immédiatement le travail de toutes les équipes. Les différentes répliques d'une information divergeront, obligeant encore une fois à allouer du temps pour réaligner les documents. Si le conflit n'est pas résolu immédiatement, l'effort nécessaire pour retrouver la source du conflit et le résoudre va être beaucoup plus important, conduisant à une perte de temps de développement très importante, d'autant plus que les différents intervenants du flot ont une vision très restreinte du flot global.

2.4.3 Les développements itératifs et les méthodes agiles

Dans ce contexte, un moyen de maintenir la cohérence est d'adopter une méthodologie qui, appliquée scrupuleusement, impose des méthodes de travail spécifiques qui assurent la

cohérence, et qu'aucun changement ne peut introduire d'incohérence dans le corpus documentaire. Plusieurs modèles de cycle de développement inspiré par le cycle en V [20] ou le modèle cascade [21] sont communément adoptés en industrie. Cependant, ces processus sont essentiellement linéaires et basés sur des processus séquentiels.

Utiliser un flot de conception purement séquentiel n'est pas raisonnable pour des systèmes complexes comme les systèmes sur puce. En effet, il est utopique d'imaginer une spécification qui définit correctement et parfaitement le système sur puce dès le départ. Définir un système sur puce, comme tous produits manufacturés, est une activité purement intellectuelle, impliquant des erreurs même avec un outillage approprié. De plus, les contraintes de design et d'implémentation peuvent imposer un remaniement des spécifications.

Plus généralement, notre expérience dans le design de systèmes sur puce, dans un contexte d'équipes distribuées, nous montre qu'utiliser un processus imposé par le cycle de développement choisi est très peu aligné au processus réellement effectué par les acteurs du flot, qui est souvent imprévisible. Dans les faits, le processus de développement est constamment en évolution et les habitudes de travail divergent rapidement d'un processus prédéfini. Il est envisageable de créer des processus outillés, forçant la cohérence en empêchant des modifications de documents non prévues. Cependant, outre le temps de développement et d'implémentation que demanderait la conception d'un tel processus, d'un point de vue sociologique l'application de ce genre de processus serait irréaliste. En effet, les différents acteurs impliqués dans le développement ne supporteraient probablement pas d'être ainsi contraints. Afin de conserver le processus prédéfini et le processus réellement appliqué cohérent, il convient donc de faire évoluer le processus tout au long du flot de conception, en le réalignant avec la réalité des faits. Cependant, cette mise à jour devrait être outillée. En effet, demander aux acteurs de mettre à jour manuellement le processus pendant un développement nécessiterait un investissement trop important de leur part.

De plus les exigences et les spécifications évoluent rapidement et souvent. Dans le secteur des semi-conducteurs, où la réactivité est une caractéristique clé, nous ne pouvons ignorer les changements dans le besoin du client et il est impossible que les exigences soient correctement capturées du premier coup. La conception des systèmes sur puce a donc migré vers des méthodes de développement agiles [22] afin d'être plus réactifs aux changements. Appliquer les méthodes agiles consiste à adopter un développement incrémental, itératif. Ces méthodes définissent quatre valeurs fondamentales : **l'interaction avec les personnes** plutôt que les processus et les outils ; des **logiciels opérationnels** plutôt qu'une documentation exhaustive ; **la collaboration avec le client** plutôt qu'une négociation contractuelle ; **la réactivité** face aux changements plutôt que le suivi d'un plan. Pour toutes ces raisons, dans de réels cas industriels, seulement les développements itératifs et agiles peuvent aider à gérer la complexité des systèmes à développer et à être plus réactif aux changements.

Cependant, les méthodes agiles présentent par essence des points faibles [23], [24] : la planification des tâches et du développement global est difficile à estimer ; il est difficile de savoir à grande échelle l'avancement des travaux car l'architecture globale est en constante évolution, il faut accepter d'avancer dans une direction qui ne figure pas nécessairement sur le plan d'origine, l'implication du client n'est pas toujours aisée, les méthodes agiles ne sont pas vraiment adaptées aux équipes trop importantes et distribuées. En effet, les méthodes agiles se basent sur des réunions fréquentes afin de maintenir la cohérence des développements des différentes équipes, or dans le cadre d'équipes distribuées ces réunions peuvent vite devenir chronophages et ralentir considérablement les développements, perdant tout l'intérêt de l'agilité.

Enfin, dans les développements complexes par équipe, chaque ingénieur a une vision limitée du flot global. Cette vision limitée du *workflow* rend impossible pour les ingénieurs qui modifient des fichiers de prévoir et d'évaluer l'impact qu'auront leurs changements sur le travail des autres, par conséquent il est impossible pour un développeur de prévenir les acteurs impactés par une modification d'un document qu'il a lui-même engendré. Ce manque de vision global est la source de l'introduction d'incohérences dans le corpus documentaire. En effet, la non-propagation des modifications apportées par les différents intervenants créée les incohérences et celles-ci peuvent alors se développer et empirer à mesure que le développement progresse, rendant alors les documents de plus en plus divergeants et de moins en moins alignés.

2.5 Analogie entre flot de conception des systèmes sur puce et systèmes distribués

Selon [25], un processus est un « Ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie ». La Figure 2-6 est une représentation simplifiée du développement d'un composant matériel (qui est un sous-ensemble du processus de conception présenté dans la section 2.2) qui suit cette définition. L'équipe responsable de l'analyse d'architecture fournit une spécification aux équipes responsables du développement matériel (niveau TLM et RTL) et du développement logiciel. Ces trois équipes itèrent entre elles en s'échangeant des documents (modèles, code source, plateformes de tests …). Lorsque les développements sont stables, l'équipe de documentation récupère des documents afin de rédiger la documentation utilisateur du composant.

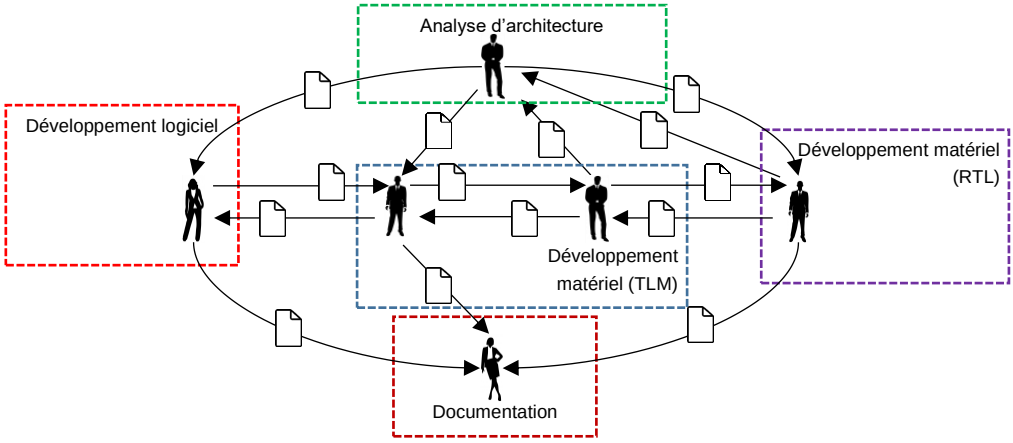


FIGURE 2-6 FLOT DE CONCEPTION D'UN COMPOSANT

2.5.1 Présentation de l'analogie

Selon [26], un système distribué est : « un **ensemble d'entités autonomes** de **calcul** (ordinateurs, PDA, processeurs, processus, processus légers etc.) [Que nous appellerons nœuds] **interconnectées** et pouvant communiquer grâce à des **échanges de messages** ». Ces systèmes, outre l'avantage de pouvoir accéder à des données stockées sur un nœud géographiquement distant, permettent d'assurer la disponibilité des données, même si un nœud vient à être déconnecté du réseau. La Figure 2-7 représente un système distribué de quatre nœuds partageant des messages.

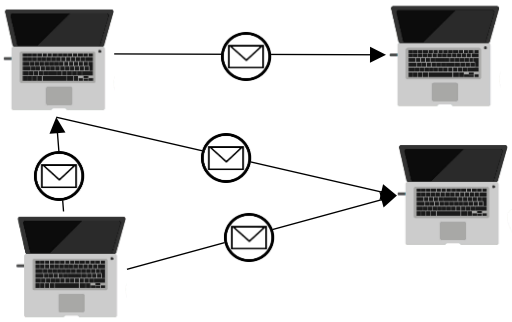


FIGURE 2-7 REPRÉSENTATION D'UN SYSTÈME DISTRIBUÉ

Il est possible de faire une analogie entre les systèmes distribués et le flot de conception des systèmes sur puce. En effet, la définition indique qu'un processus est constitué d'**activités** prenant des documents en **entrée** et produisant des documents en **sortie**. Nous pouvons donc en déduire, que ces activités (effectuée par des **équipes**) effectuent **des modifications** sur les documents reçus ou **créent** à partir de ces documents reçus (en dupliquant de l'information sémantique) de nouveaux documents avant de les **envoyer** en entrée d'une autre activité.

Nous pouvons alors voir le flot de conception comme des systèmes distribués, où les nœuds sont représentés par les équipes (ou même les employés), les messages échangés entre

les nœuds sont les documents échangés par les équipes et les modifications ou créations de documents comme les opérations (ou calculs) effectuées sur les nœuds.

La cohérence des données est un domaine de recherche connu des systèmes distribués. Cette analogie nous permet de transposer ces connaissances issues des systèmes distribués vers le flot de conception des systèmes sur puce.

2.5.2 Modèles de cohérence appliqués au flot de conception

En informatique, la cohérence des données est la capacité pour un système à refléter sur la copie d'une donnée les modifications intervenues sur d'autres copies de cette donnée [27]. En adaptant cette définition selon l'analogie présentée, la cohérence du flot de conception des systèmes sur puce est la capacité à propager une modification intervenue sur une information sémantique contenue dans un document vers toutes les copies de cette même information sémantique dans les autres documents.

La notion centrale est le modèle de cohérence utilisé. Un modèle de cohérence s'affiche comme un contrat passé entre le système et l'utilisateur. Il définit les critères déterminant la valeur retournée par une lecture en fonction des écritures précédentes. Il existe plusieurs modèles de cohérence appartenant aux classes de cohérence forte ou faible [27], [28]. Nous pouvons résumer ces deux classes de modèles de cohérence par les définitions suivantes :

- La cohérence forte garantit que toute lecture d'une copie d'une donnée reflétera **instantanément** toute modification antérieure à la lecture intervenue sur n'importe quelle copie de la donnée.
- La cohérence faible assure que si une copie est modifiée, toutes les copies des données reflèteront ces modifications **au bout d'un certain temps,** mais la modification n'est pas forcément immédiate.

Les modèles de cohérence forte ont été introduits les premiers, dans le domaine de l'informatique, et facilitent la vision des données fournie à l'utilisateur. Ils imposent des contraintes importantes entre la dernière lecture et la prochaine écriture d'une donnée. Cette stratégie entraine l'implémentation de protocoles complexes et extrêmement couteux en communications réseau, ou en réunion et communication inter-équipe dans notre cas. Afin de gagner en efficacité, plusieurs modèles de cohérence ont donnés lieu à différents protocoles de cohérence.

2.5.2.1 Modèle de cohérence forte

Les modèles de cohérence forte sont caractérisés par des contraintes fortes entre la dernière écriture et la prochaine lecture.

- ***Atomic consistency*** est un modèle idéal où chaque lecture rend la dernière valeur écrite dans la donnée.

- **Analogie :** Ce modèle de cohérence implique que chaque modification d'une information sémantique dans un document est instantanément et automatiquement propagée à tous les documents utilisant la même information sémantique. Ce modèle est inapplicable dans le flot de conception des systèmes sur puce. En effet, il se base sur l'existence d'une source d'information unique et commune regroupant toutes les informations sémantiques et suppose la possibilité de générer à partir de cette base tous les formats possibles utilisés dans le flot.

- ***Immediate consistency*** assure que l'écriture d'une donnée est finalisée seulement lorsque tous les nœuds ont été synchronisés [29].
  - **Analogie :** Ce modèle implique qu'une modification d'une information sémantique est estampillée terminé lorsque toutes les répliques de cette information sémantique ont été modifiées également. Ce modèle est celui que l'on peut retrouver dans les cycles de développement séquentiel (type modèle cascade). En effet, une modification d'un document venant forcément du haut du flot, la propagation va se faire à tous les documents en aval avant que celle-ci soit terminée. Ce modèle très rigide aux changements est trop couteux en temps de développement et de propagation de changement pour être applicable dans notre cas.

2.5.2.2 Modèle de cohérence faible

Les modèles de cohérence faible ont été introduits afin de diminuer le nombre d'échanges réseau induit par les protocoles de cohérence forte. Ils tirent parti du fait que les applications distribuées imposent déjà un ordre sur les accès mémoire par l'utilisation explicite de mécanismes de synchronisation. La technique employée consiste à retarder les mises à jour jusqu'aux prochains points de synchronisation, diminuant ainsi le nombre de communications, ce qui améliore les performances générales.

- ***Weak consistency*** fait la distinction entre les accès ordinaires à la mémoire et les accès synchronisés. Seuls les accès synchronisés garantissent la cohérence de la mémoire partagée par l'utilisation d'objets de synchronisation comme les verrous ou les barrières. Ce modèle garantit que la mémoire est cohérente à chacun des points de synchronisation pendant lesquels toutes les informations sont mises à jour.
  - **Analogie :** Ce modèle implique de bloquer les modifications d'informations sémantiques au moyen de points de synchronisation. La modification des informations sémantiques est bloquée tant qu'un autre acteur modifie une information sémantique ou que le système propage une modification et assure la cohérence du flot. Pour des raisons de temps de développement, ce modèle ne peut pas être appliqué dans le flot de conception des systèmes sur puce.

- ***Eager realease consistency*** (ERC) améliore la *weak consistency* en ne mettant à jour que les données modifiées entre deux synchronisations. La cohérence de la mémoire est assurée par la propagation vers les autres nœuds des modifications effectuées sur la donnée dans une section critique. La particularité de ce modèle est que la mise à jour intervient lors de l'appel à la commande de **libération**. Ce protocole génère des communications inutiles vers des nœuds n'effectuant par la suite aucun accès sur les données mises à jour. Le protocole associé est mis en œuvre dans le système à mémoire virtuellement partagée de Munin [30].
    - **Analogie :** Ce modèle implique que la propagation de modifications d'une information sémantique n'est propagée que vers les documents contenant cette même information sémantique, ce qui peut être difficile à identifier. De plus, tout comme le modèle *weak consistency*, la non disponibilité des informations pendant les phases de propagation n'est pas applicable dans le flot de conception des systèmes sur puce pour des raisons d'allongement du temps de propagation.
- ***Writes-follows-reads consistency*** permet de gérer les incohérences en garantissant que les écritures d'un nœud soient propagées après la réalisation d'une opération de lecture.
    - **Analogie :** Ce modèle suggère que la propagation d'une modification d'une information sémantique a lieu lorsqu'un acteur modifie un document partageant l'information sémantique après la lecture de la première modification. Ce modèle laisse donc les incohérences entre document existées tant que les acteurs n'ont pas vus une modification.
- ***Eventual consistency*** demande simplement que les écritures d'un nœud soient vues par les autres au bout d'une période donnée, sans contrainte d'ordre. Il s'agit du modèle de cohérence le plus faible.
    - **Analogie :** Ce modèle de cohérence impose que les modifications d'une information sémantique soient propagées au bout d'un certain temps, non spécifié.

2.5.3 Analyse du modèle de cohérence du flot de conception

Cette présentation de différents modèles de cohérence laisse apparaitre un problème théorisé par Brewer [31]. Le théorème du CAP (*Consistency Avaibility & Partition tolerence*) définit qu'il est impossible pour un système distribué de fournir à la fois de la cohérence (si une information sémantique est modifiée dans le corpus de documents par un acteur, et que l'on demande ensuite à un autre acteur l'information, sa réponse doit correspondre au changement fait par le premier acteur), de la disponibilité (lorsque l'on sollicite un acteur pour obtenir une information sémantique, il répond) et une tolérance au partitionnement (si une portion de l'équipe ne peut pas communiquer avec une autre, le développement du système continue correctement) ; un compromis doit être fait. Le

théorème PACELC [32] complète cette analyse en précisant que même en l'absence de partitionnement, il faudra faire un compromis entre la cohérence et la latence.

Nous pouvons utiliser l'analogie présentée afin d'appliquer ces théorèmes au flot de conception de systèmes sur puce. Alors que les cycles de développement standards (comme le cycle en V et le modèle en cascade) favorisent la cohérence, ils sacrifient la latence. Il s'agit ici des systèmes de blocage des modèles de cohérence forte. En effet, les processus séquentiels imposés par ces cycles de développement forcent une modification à se propager du haut du flot vers le bas, étape après étape. Au cours de cette propagation, toutes les autres équipes sont à l'arrêt, attendant d'avoir accès à la donnée corrigée.

Au contraire, les méthodes agiles reposent sur une bonne disponibilité et une communication très régulière (donc une bonne tolérance au partitionnement). Elles tendent à favoriser la réactivité (communication quotidienne et itérations courtes). Cependant, à partir d'une certaine échelle, la cohérence du corpus documentaire est sacrifiée. En effet, les différentes équipes se concentrant sur des problèmes particuliers ne peuvent pas assurer la cohérence avec les autres parties développées par les autres équipes car la communication n'est plus aussi facile et efficace. Une forme de partitionnement se créé. Ce problème peut même s'étendre aux différents développeurs au sein d'une même équipe. La maintenance de cohérence avec les méthodes agiles se base sur des réunions fréquentes entre les équipes afin de se synchroniser. Or, ces réunions ne sont pas réalisables au sein des très grandes entreprises comme STMicroelectronics où les équipes sont mondialement distribuées. De plus, le manque de vision global des différents intervenants peut les empêcher de se synchroniser avec toutes les personnes concernées.

Cet état de fait laisse entendre que le modèle de cohérence appliqué **implicitement** lors de l'utilisation des méthodes agiles est l'*Eventual consistency*. En effet, la cohérence du corpus de document n'est assurée qu'au bout d'un certain temps, non défini, suite à la propagation manuelle des modifications des informations sémantiques par les acteurs du flot.

2.6 Conclusion

Dans ce chapitre, nous avons présenté le contexte de notre travail. Le développement des systèmes sur puce est une opération complexe. En effet, de nombreuses étapes sont nécessaires à son développement. Chacune de ces étapes fait appel à plusieurs équipes dédiées à la représentation d'un aspect du système sur puce. Ces différentes équipes ont besoin de plusieurs langages et outils spécifiques à leur cœur de métier afin d'exprimer leurs idées sur une vue particulière du système. Malheureusement, la maintenance de cohérence du corpus documentaire est une source de problèmes grandissante, pouvant causer un allongement du temps de développement afin de réconcilier des documents incohérents.

Nous proposons également une analogie entre les systèmes distribués et le flot de conception des systèmes sur puce. Cette analogie nous permet de rapprocher les modèles de

cohérence au flot de conception. La Table 2-3 donne un résumé des différents modèles de cohérence que nous avons présenté, ainsi que leurs avantages et inconvénients pour leurs applications dans le flot de conception des systèmes sur puce

TABLE 2-3 RÉSUMÉ DES AVANTAGES ET INCONVÉNIENTS DES MODÈLES DE COHÉRENCE APPLIQUÉS AU FLOT DE CONCEPTION DES SYSTÈMES SUR PUCE

Forte	*Atomic*	• Cohérence	• Inapplicable
Forte	*Immediate*	• Cohérence (une modification non terminée tant qu'elle n'est pas propagée)	• Disponibilité • Tolérance au partitionnement
Faible	*Weak*	• Cohérence (par système de synchronisation)	• Disponibilité
Faible	*Eager release*	• Cohérence (par système de synchronisation) • Cible précisément l'impact d'une modification	• Disponibilité
Faible	*Write-follows-read*	• Disponibilité • Laisse les incohérences exister	• Cohérence
Faible	*Eventual*	• Réactivité	• Cohérence

Dans notre contexte, il est impossible d'appliquer une méthode de cohérence forte dans la description d'architecture. En effet, vouloir une cohérence forte impliquerait, tout d'abord, d'être capable à tout instant de propager une modification sémantique aux documents exploitant la même information sémantique, ce qui est impossible pour nous, car nous restons limité par les différentes syntaxes. Le deuxième frein à l'utilisation d'une méthode de cohérence forte (qui bloque également l'utilisation de méthode de cohérence faible utilisant des systèmes de synchronisation) réside dans le fait que les systèmes sur puce doivent être développés le plus rapidement possible afin de gagner le plus de part de marché possible. Dans ce but, les divers développements sont fractionnés et distribués à différentes équipes, travaillant en parallèle. Cette méthode de travail est incompatible avec les systèmes de synchronisation des méthodes de cohérence faible ou d'attente que nécessitent les méthodes de cohérence forte. Aujourd'hui, implicitement le modèle de cohérence *Eventually consistency* est appliqué, garantissant la cohérence au bout d'un temps, pouvant être potentiellement infini.

Chapitre 3
Gestion des sources d'incohérences dans le flot de conception des systèmes sur puce

Sommaire

La maintenance de cohérence du corpus documentaire impliqué dans le développement d'un système sur puce peut être rendue plus compliquée par les spécificités du flot de conception. En effet, l'hétérogénéité des formats, la distribution des équipes et la parallélisation des développements cause une difficulté majeure aux intervenants du flot pour prévoir l'impact d'une modification qu'ils apportent à un document et pour propager cette modification. De plus, ces contraintes ne peuvent être effacées ou négligées, elles font partie intégrante du flot de conception de systèmes complexes.

Dans ce chapitre, nous présenterons tout d'abord des outils collaboratifs utilisés afin de permettre une meilleure communication et de meilleurs échanges entre les équipes et individus, nécessaires pour gérer le nombre de développement et la distribution des équipes. Puis nous développerons l'utilisation d'outils de centralisation de l'information permettant l'interopérabilité entre les différents outils nécessaires au développement de systèmes sur puce. Ces outils de centralisation proposent d'alléger le poids de l'hétérogénéité des formats et de la distribution des équipes. Enfin, nous aborderons le standard ISO/IEC/IEEE 42010 permettant de décrire l'architecture du flot de conception des systèmes sur puce, dans le but de rendre tous les acteurs d'un développement conscients du processus global. La Figure 3-1 représente une carte heuristique des outils utilisés au sein de STMicroelectronics pour gérer les spécificités du flot de conception et permet de se représenter l'utilité de chacun des outils présenté dans ce chapitre.

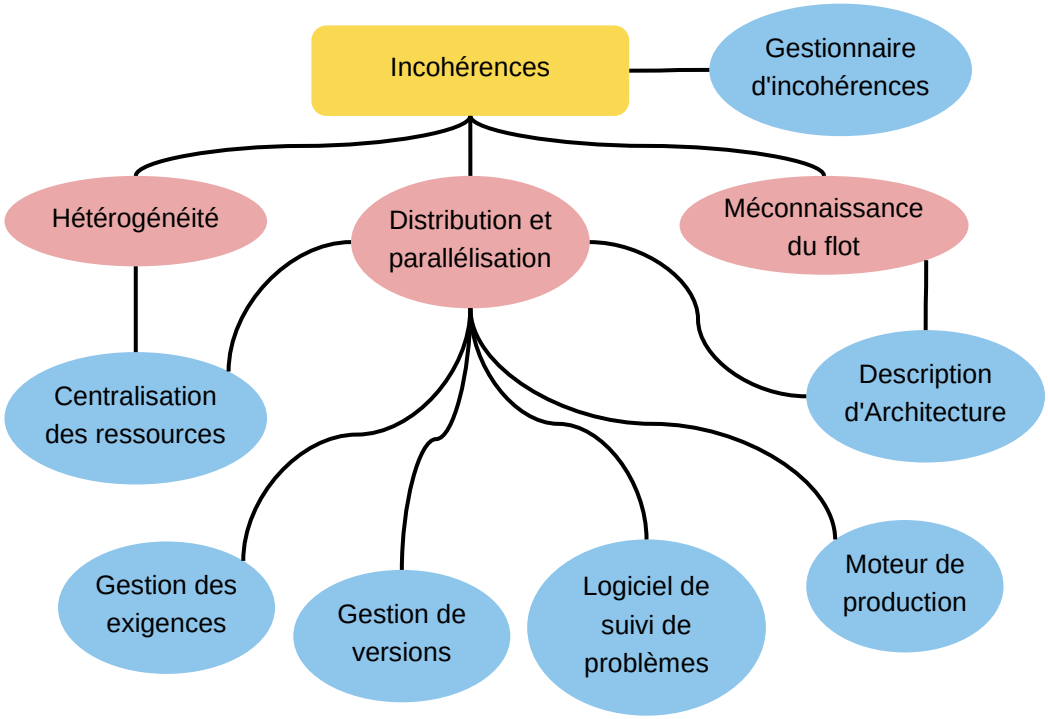


FIGURE 3-1 CARTE HEURISTIQUE DES OUTILS UTILISÉS À STMICROELECTRONICS POUR GÉRER LES SPÉCIFICITÉS DU FLOT

3.1 Outils collaboratifs

Le développement des systèmes sur puce fait appel à de nombreux intervenants. STMicroelectronics est une société multi sites ; le développement se fait donc grâce à des équipes distribuées géographiquement autour du monde. Des outils collaboratifs permettent de maximiser l'efficience d'un groupe associés à des projets d'envergure.

3.1.1 Gestion des exigences

La gestion des exigences consiste à hiérarchiser les exigences d'un projet, à détecter les incohérences entre elles et à assurer leurs traçabilités et leurs applications.

De nombreux cadres normatifs [33], [34] imposent une gestion des exigences et donne lieu à une quantité de documents dont la cohérence et la qualité conditionnent le succès ou l'échec des projets. Il existe des logiciels spécialisés qui permettent d'aider à la réalisation de cette tâche (comme Reqtify [35], Doors [36]).

La traçabilité des exigences est une sous branche et un concept clé mis en avant de la gestion des exigences dans le domaine du développement logiciel et de l'ingénierie systèmes. Selon [37], la traçabilité d'exigence est la capacité de suivre la vie d'une exigence, dans les deux directions (amont et aval) d'un projet, c'est-à-dire, de ses origines en passant par son développement et sa spécification, à son déploiement et son utilisation dans les phases suivantes tout en considérant les périodes d'amélioration et d'itération émanant d'autres phases (conception, construction, validation …). La traçabilité des exigences consiste donc à documenter la vie des exigences. Il doit être possible de retracer jusqu'à leur origine chacune des exigences et chacun des changements les affectant ; les exigences doivent donc être documentées pour achever la traçabilité. Les exigences proviennent souvent de sources diverses, telles que l'équipe marketing, les clients ou les utilisateurs. Grâce à la traçabilité des exigences, chaque fonctionnalité implémentée pourra être reliée à une personne ou à un groupe l'ayant demandée durant la phase de définition des exigences.

Une gestion des exigences appliquée et suivie scrupuleusement tout au long du développement du système peut aider à maintenir la cohérence. Cependant, dans la majorité des cas, les exigences ne sont tracées qu'entre l'implémentation effective de l'exigence et l'expression de l'exigence elle-même. Les documents dérivés contenant des informations sémantiques communes ne seront pas tracés.

3.1.2 Gestion de versions

Un logiciel de gestion de versions (Git, Mercurial, Subversion) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes. Il agit sur une arborescence de fichiers afin de conserver toutes les versions des fichiers, ainsi que les différences entre eux. Ce système permet par exemple de mutualiser un développement. Le système travaille par fusion de copies locales et distantes, et non par écrasement de la version distante par la version locale.

Les logiciels de gestion de versions permettent de facilement partager des documents et des modifications de documents. Ils sont indissociables du développement des systèmes sur puce. En effet, dès qu'un acteur du flot termine la création d'un document, celui-ci publie ce document dans un *dépôt* (serveur des logiciels de gestion de version). La publication du

document dans le *dépôt* est une étape obligatoire et inscrite dans les pratiques de travail classiques.

Certains logiciels de gestion de version sont associés à des mécanismes qui préviennent, d'une modification d'un document, des personnes inscrites dans une liste de diffusion. Ces mécanismes permettent donc de dire quel fichier est modifié. En revanche, ils ne vont pas prévenir quels sont les documents partageant de l'information et donc impactés par ce changement. Le développeur doit faire un effort mental pour savoir quel fichier est impacté. La cohérence, tel que définie précédemment, n'est donc ni assistée, ni garantie par ces systèmes.

3.1.3 Moteur de production

Un moteur de production est un outil dont la fonction principale consiste à automatiser, ordonnancer et piloter l'ensemble des actions (préprocessing, compilation, éditions des liens, etc.) contribuant, à partir de données sources, à la production d'un ensemble opérationnel.

Les moteurs tendent désormais à être directement intégrés aux environnements de développement (comme Eclipse [38], [39], Visual Studio [40]). Ces outils permettent de générer automatiquement des documents. Ces documents sont donc cohérents, à la génération, entre eux. Cependant, la cohérence n'est assurée qu'entre les documents en entrée et en sortie du moteur de production. De plus, tous les moteurs de production ne sont pas exécutés à la volée, une modification d'une source ou du document généré est donc toujours possible, rendant les documents incohérents.

3.1.4 Logiciel de suivi de problèmes

Un logiciel de suivi de problèmes permet aux utilisateurs et aux développeurs d'améliorer la qualité d'un système. Les utilisateurs soumettent leurs demandes d'assistance dans l'outil. Les développeurs sont alors toujours au fait des problèmes rencontrés.

La plupart des systèmes de suivi de problèmes permettent aux utilisateurs de rentrer directement les problèmes rencontrés au niveau technique, respect des règles de gestion fonctionnelles, mais également des demandes d'amélioration ou de nouvelles fonctionnalités. À l'origine, ce type de logiciel était conçu pour suivre les incidents ou anomalies d'un projet. Désormais, certains logiciels de suivi de problèmes sont configurables et permettent de gérer tous types de tickets ou artefacts : tâches, demandes de support, exigences, contacts.

Cet outil permet aux utilisateurs de signaler une incohérence entre les besoins du client (qui peuvent être informels et implicites ou formellement définis et explicites) et l'implémentation concrète du système. Cependant, la notion de cohérence est basée sur l'humain. En effet, si un acteur du flot s'aperçoit d'une incohérence, un effort devra être fait de sa part afin de notifier (en ouvrant un ticket) le problème. La correction de l'incohérence

n'est pas garantie, pour différentes raisons (financière ou temporelle principalement) le développeur peut choisir de ne pas corriger l'incohérence.

3.1.5 Outils de centralisation des ressources

Plusieurs outils ont été développés afin de couvrir à la fois la distribution des équipes (donc en favorisant le travail collaboratif) et la gestion des différents outils utilisés dans un flot de conception. Nous présentons dans cette section deux exemples de ces outils : Magillem Content Platform et ModelBus.

3.1.5.1 Magillem Content Platform

Magillem Content Platform (ou MCP) [41] est une suite de logiciels permettant de fragmenter des ressources documentaires, de créer des publications à partir des fragments, d'identifier des fragments impactés par l'évolution d'un référentiel et de consolider les fragments documentaires lorsque des modifications sur un référentiel ont été détectés de façon automatique ou assistée. La Figure 3-2 (extraite de [42]) montre le fonctionnement et l'utilisation de MCP. La partie de gauche montre la création et le stockage des fragments à partir de documents MS Word, Framemaker ou DITA, qui sont ensuite convertis et stockés dans un format propriétaire. La partie de droite illustre l'utilisation de ces fragments afin de créer de nouveaux documents et la possibilité de les exporter en PDF, MS Word, DITA, Framemaker ou HTML.

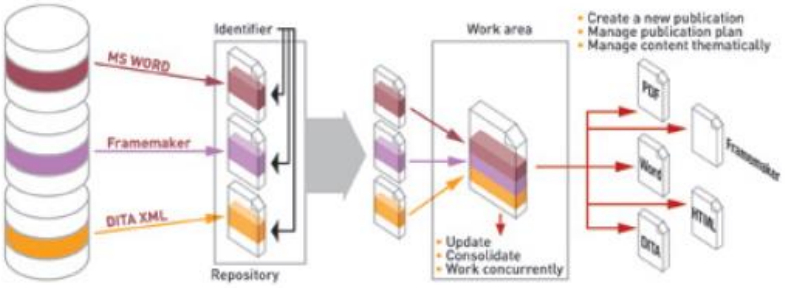


FIGURE 3-2 UTILISATION DE MAGILLEM CONTENT PLATFORM

Ce logiciel présente l'avantage de maintenir la cohérence entre plusieurs formats en centralisant les informations sémantiques sous une forme unique. Cependant, le nombre de formats traités est très limité. En effet, MCP est surtout utilisé dans le but de créer ou de maintenir des documentations (ou des documents juridiques), ces principales sources d'informations sémantiques sont donc naturellement des documents en langage naturel. L'extension de MCP pour traiter tous les formats de documents impliqués dans le flot de conception des systèmes sur puce serait réalisable en échange d'un très grand investissement en termes de temps de développement. Il faudrait pour chaque format de document être capable d'importer et de fragmenter ces documents. De plus, nous ne savons pas si le format propriétaire utilisé pour formaliser les fragments permettrait de stocker les informations sémantiques de tous les formats.

3.1.5.2 OSLC et Modelbus

ModelBus [43] peut être vue comme une généralisation du principe appliqué dans MCP. ModelBus est un framework permettant de gérer des processus de développement complexes intégrant des outils hétérogènes. Il permet d'intégrer des outils de différents fournisseurs utilisés à des fins différentes. La Figure 3-3 (extraite de [44]) montre le fonctionnement et le positionnement de ModelBus. Les outils connectés à ModelBus offrent ou consomment des services agissant sur des informations. Lorsqu'une information est créée, lors du développement d'un système ou d'un logiciel, elle est mise à disposition dans le « bus virtuel » qu'est ModelBus. Cette information est alors accessible par les autres outils.

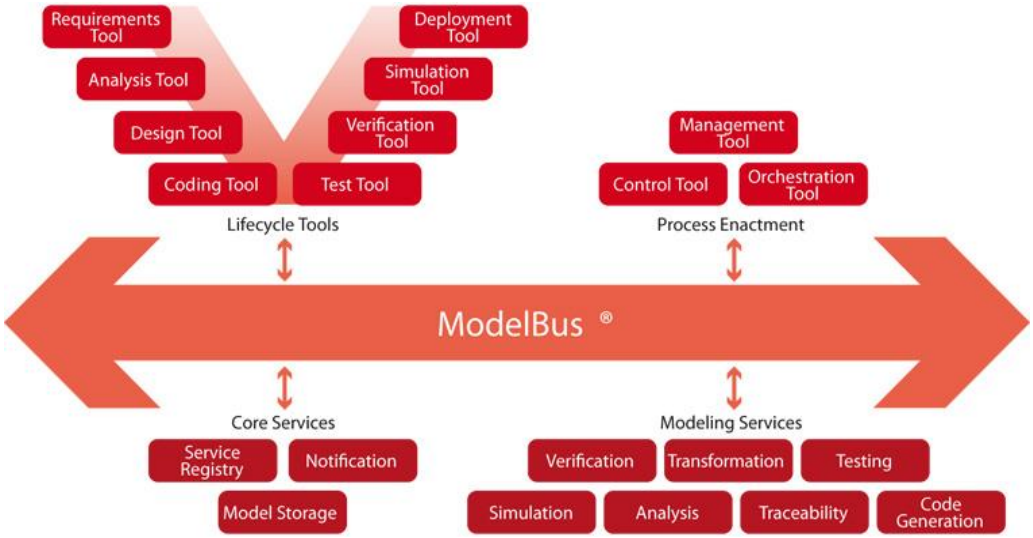


FIGURE 3-3 MODELBUS

L'interopérabilité d'outils hétérogènes que présente ModelBus suit les spécifications créent par la communauté *Open Services for Lifecycle Collaboration* [45] (OSLC). Ces spécifications visent à intégrer et partager les données produites par les outils utilisés dans le flot de conception d'un système ou d'un logiciel.

Bien que le nombre de formats et d'outils pouvant être analysés et utilisés par ModelBus (Eclipse Papyrus, Sparx Enterprise Architect [46], Matlab Simulink, Microsoft Office, Doors, Trac – Issue tracker [47]) soit plus important qu'avec MCP, la même limitation s'impose. En effet, la variété d'outils et de formats utilisés dans le flot de conception des systèmes sur puce obligerait à développer tous les adaptateurs permettant d'importer les ressources dans ModelBus. De plus, des outils propriétaires pourraient empêcher l'utilisation de Modelbus sur l'intégralité du flot de conception, leurs formats utilisés étant cryptés. Nous pensons que la granularité de l'information, c'est-à-dire la taille du fragment d'information traitable, utilisée, tant par MCP que par Modelbus, est beaucoup trop fine pour permettre une adoption universelle de ces techniques. En effet, l'accès à chaque fragment d'information contenu dans différents documents oblige à avoir la capacité d'analyser et d'extraire ces informations. Cependant, la variété d'outils et de langages ou formalismes

utilisés ne permet pas forcement d'avoir accès à ces informations. La manipulation de l'ensemble des documents impliqués dans le flot de conception des systèmes sur puce nécessite donc l'utilisation d'une granularité plus grosse afin d'éviter l'analyse du contenu des documents.

3.2 Gestionnaire d'incohérence (*inconsistency checking*)

De nombreux outils ont été développés pour aider à maintenir automatiquement la cohérence d'un ensemble de document. C'est le cas des gestionnaires d'incohérences, qui permettent de vérifier la cohérence de modèles en suivant des règles de vérification.

Dans les années 2000, une première génération de gestionnaire d'incohérences a émergée[48], [49]. Ces gestionnaires d'incohérences sont basés sur des règles de maintien de cohérence et exécutés en batch. Cette première génération présente des performances insuffisantes dues au fait que lorsqu'un document est modifié toutes les règles doivent être vérifiées à nouveau. Ces exécutions incessantes causent une grande perte de temps. Comme mentionné dans [50], vérifier un « gros » modèle avec ces gestionnaires d'incohérences peut prendre plusieurs heures d'exécution.

Dans [51], Xavier Blanc *et al.* proposent une nouvelle méthodes de gestion d'incohérence. Les modèles sont représentés comme une séquence d'opérations élémentaires permettant d'arriver à la version courante. Cette méthode s'oppose à une représentation plus classique où les modèles sont représentés par un ensemble d'éléments contenu dans celui-ci.

Cette représentation permet de vérifier la cohérence de l'ensemble des modèles incrémentalement, en analysant et traçant la modification appliquée. Cependant, les règles de cohérence sont définies explicitement pour chaque type de modifications appliquées. Cette méthode requiert un effort manuel supplémentaire de la part des ingénieurs, qui peut être à la fois une nouvelle source d'erreur, et susceptible d'être négligé par les développeurs, comme les tests ou la documentation par exemple [52], [53].

Dans [50] Alexander Egyed propose une approche automatisée permettant d'automatiser la génération des règles. L'approche suggère d'observer le comportement des règles de cohérence afin de comprendre comment celle-ci impacte le modèle. Les règles de cohérence ne sont pas écrites dans un langage spécifique, ce qui peut augmenter les chances d'adoption de la méthode par les développeurs.

Les gestionnaires d'incohérence tels qu'ils sont aujourd'hui étudiés dans le monde académique soulèvent un problème majeur pour leur adoption dans le domaine des semi-conducteurs. Ces méthodes sont développées et fonctionnent pour une utilisation sur des modèles. Cependant, les différentes tâches qui composent le développement d'un système sur puce font appel à plusieurs formalismes différents. Tous ces formalismes ne sont pas des

modèles formellement définis. Les méthodes de contrôle d'incohérences ne peuvent donc pas être appliquées sur l'ensemble de documents nécessaires au développement de systèmes complexes. Cependant, la représentation des modèles comme étant des séquences d'opérations peut s'appliquer également à tous documents. En effet, il est possible de voir les documents comme une suite d'ajouts, de suppressions, ou de modifications d'information sémantique. Cette vue rapproche d'ailleurs le document des gestionnaires de version, qui eux même sont capables de tracer l'évolution d'un document.

3.3 Description d'architecture

Comme nous l'avons dit dans le chapitre précédent, une des causes de la mauvaise propagation des changements vient d'une méconnaissance du flot global de la part des différents acteurs de ce flot. En effet, lorsqu'un développeur modifie un document, il ne sait pas exactement quelle est la portée de sa modification. Il n'est donc pas en mesure de propager convenablement son changement en prévenant tous les acteurs impactés. La modélisation de la description de l'architecture peut aider à résoudre ce problème. En décrivant les différentes équipes, leurs cœurs de métier et leurs interactions il est possible de rendre tous les acteurs conscients du processus global.

« Tous les systèmes ont une architecture » [54] et décrire cette architecture est le but d'un développement. Lorsque le nombre d'intervenants impliqués dans le développement d'un système et que le nombre de documents traités par chaque intervenant n'est pas excessif, l'architecture du système peut facilement être décrite.

Cependant, dans notre contexte, le développement n'est jamais effectué par une seule et même personne. En effet, plusieurs intervenants participent et chacun d'eux contribue avec son expertise sur un aspect particulier du système. À travers le processus, ces intervenants produisent des documents (comme du code source, des graphiques, des schémas de bases de données, des documents textuels, des documentations, des diagrammes, des spécifications, des modèles de designs, des captures d'écran …). C'est la somme de tous ces documents qui décrivent le système sur puce et des étapes permettant la production de ces documents qui constitue la description d'architecture du système.

La description de l'architecture est un élément fondamental pour le développement de systèmes complexes. La possibilité d'exprimer, d'analyser et de communiquer ces architectures est une clé du succès [55].

Le standard ISO/IEC/IEEE 42010 :2011, *Systems and software engineering Architecture description* [2] donne une vision structurée et organisée des différents concepts nécessaires à la description d'architecture, comme les acteurs du système, les préoccupations des acteurs, les vues architecturales, les structures d'architecture. Ce standard définit également les meilleures pratiques pour définir des descriptions d'architectures et pour maximiser leur utilité tout au long du cycle de développement du système. L'approche que nous proposons,

présentée dans le chapitre suivant, se base sur ce standard pour capturer formellement certains aspects dynamiques de la description d'architecture des systèmes sur puce afin de garantir une meilleure cohérence de celle-ci.

3.3.1 Aperçu du standard ISO/IEC/IEEE 42010

En utilisation depuis sa standardisation par l'IEEE en 2000, le standard ISO/IEC/IEEE 42010 a été approuvé en tant que norme révisée par le conseil de l'IEEE – SA Standards le 31 octobre 2011. Alors que l'édition de 2000 portait sur les propriétés des descriptions d'architecture individuelles, comme la définition de ce qu'est une description d'architecture (AD) complète, cohérente etc… La révision actuelle apporte des définitions quant aux mécanismes de mise au point pour la réutilisation et l'interopérabilité des architectures techniques par le biais de trois mécanismes :

- Les réflexions d'architecture (ou *Architecture Reflexion*) : manières communes d'exprimer et de résoudre un ensemble de préoccupations architecturales connues qui peut être réutilisé dans des projets.
- Les Langages de description d'architecture (ou *Architecture Description Language*, ADL) : langages spéciales capables d'exprimer certaines préoccupations du système grâce à une ou plusieurs ressources de modélisation.
- Les structures d'architecture (ou *Architecture Framework*) : ensemble de points de vue coordonné pour l'utilisation par un intervenant particulier ou pour un domaine d'application particulier.

Les points de vue architecturaux, tels que définis par le standard, codifient la description d'architecture en spécifiant une description architecture comme un ensemble de vues différentes de l'architecture totale. Chaque vue est créée en utilisant des conventions, notations et pratiques de modélisations spécifiques. Cette idée date des années 1970 et apparait également dans l'ingénierie des exigences [56].

L'idée principale d'un point de vue architectural est un ensemble dirigé de ressources de modélisation capable de répondre à un ensemble de préoccupations pour un ou des acteurs spécifiques. En tant que tel, un point de vue est une forme de connaissance architecturale réutilisable (comme un patron, ou un IP block) utilisé pour résoudre un problème de description d'architecture.

3.3.2 Description d'Architecture, acteurs et préoccupations

La Figure 3-4 (extraite de [2]) fournit un fragment de l'ontologie définie dans ISO/IEC/IEEE 42010 sur le contexte de création et d'utilisation d'une description d'architecture. Le concept de *System* représente tout ce qui est d'intérêt lors de la création de l'architecture. Le *System* peut représenter des parties de logiciel, des composants matériels, des êtres humains, processus, matériels… La nature du système n'est pas définie par le standard. Dans le contexte de cette dissertation, nous considérons le système comme

le flot de conception des SoC que tous les acteurs cherchent à développer ou le flot de conception d'une partie du système (bloc de propriété intellectuelle). Un système est situé dans un *Environment,* qui est tout ce qui peut influer sur le fonctionnement du système. L'environnement du système peut contenir d'autres systèmes.

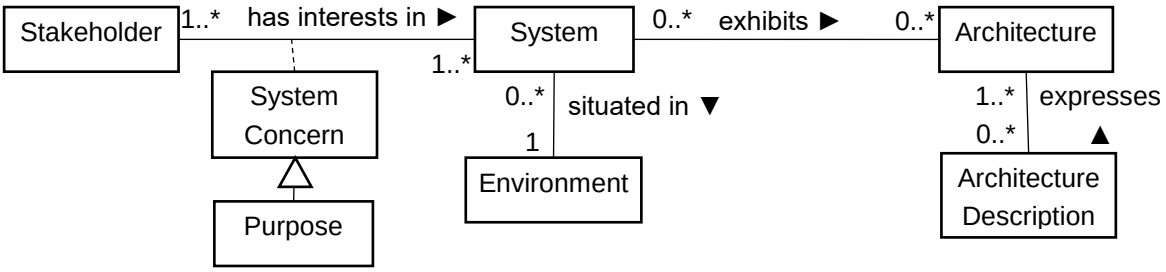


FIGURE 3-4 CONTEXTE D'UNE DESCRIPTION D'ARCHITECTURE

Comme décrit dans ISO/IEC/IEEE 42010, les acteurs (ou *stakeholders)* du système sont des éléments ayant un intérêt sur le système. Un acteur peut être vu comme un individu ayant un lien avec le système ; par exemple le terme acteur englobe les architectes, les commerciaux, les développeurs, les designers, les testeurs etc …

Les intérêts (et attentes) qu'un acteur peut avoir sur le système sont définis comme des préoccupations (ou *Concerns*). Les préoccupations peuvent aller de la faisabilité, les limitations connues, la structure, le comportement, les performances, l'utilisation des ressources, la sécurité jusqu'aux buts et stratégies commerciales. Les acteurs peuvent définir et assigner divers buts (ou *purpose*) à un système, qui sont un type de préoccupations spéciales.

L'architecture d'un système constitue ce qui est essentiel à propos du système considéré en relation avec son environnement. Il est important de noter que le standard ISO/IEC/IEEE 42010 fait une distinction entre ce qu'est une architecture d'un système et ce qu'est une description d'architecture : fondamentalement, tandis qu'une description d'architecture est une production concrète, une architecture est abstraite, qui consiste en un ensemble de concepts et de propriétés. Donc, une AD est une production utilisée pour exprimer l'architecture du système d'intérêt. Dépendant des préoccupations de l'acteur, la description d'architecture peut être utilisée pour différent buts : pour le développement et la conception du système, pour l'analyse de données, pour documenter les aspects essentiels d'un système, pour budgétiser une activité, et récemment pour les activités de tests [57].

3.3.3 Vue et Point de vue architectural

Un concept très important du standard ISO/IEC/IEEE 42010 est que la description d'architecture est composée de vues et de points de vue architecturaux (*View* et *Viewpoint*). Sous cette perspective, une description d'architecture n'est pas un artefact monolithique ; il peut être vu comme un ensemble coordonné de vue et de point de vue, chacun d'eux se focalisant sur un aspect spécifique du système en développement. Plus spécifiquement,

comme montré dans la Figure 3-5 (extraite de [2]), chaque vue est définie pour adresser un ensemble de préoccupations spécifiques (qui sont également liées par un ensemble spécifique d'acteurs).

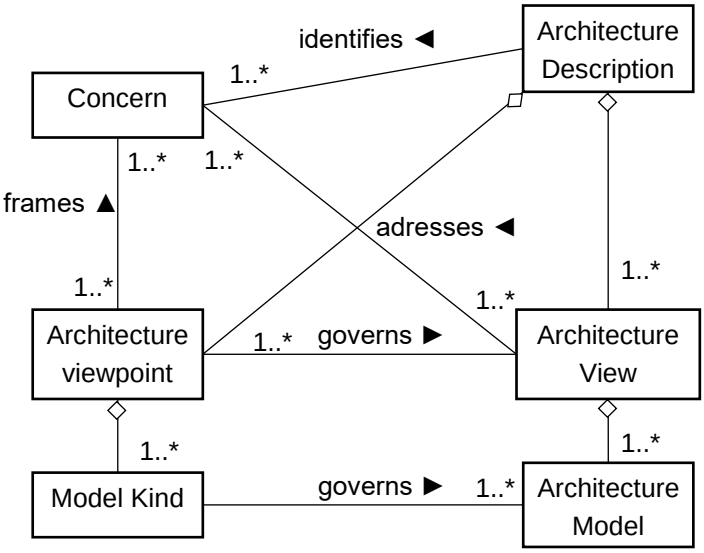


FIGURE 3-5 VUES ET POINTS DE VUE ARCHITECTURAUX

Le fait que la description d'architecture soit faite de différentes vues aide à gérer la complexité inhérente du système en développement. Par exemple, les architectes logiciels peuvent spécifier une vue purement structurelle pour décrire les différents composants du système, tandis qu'une autre vue se focalise sur le processus de développement, et une autre vue sur la sécurité.

Une vue architecturale exprime l'architecture du système d'intérêt en accord avec un point de vue architectural. Un point de vue gouverne sa vue correspondante, plus spécifiquement : le point de vue établit les conventions de construction, d'interprétation et d'analyse de la vue pour adresser les préoccupations cadrées par le point de vue. Les conventions de points de vue peuvent inclure les langages, les notations, les types de modèles, les règles de design, les méthodes de modélisation, les techniques d'analyse. Un point de vue peut être : structurel, comportemental, financier, informatique, technologique.

Les vues et points de vue sont composés respectivement d'un ou plusieurs modèles d'architecture et de types de modèle. Les modèles d'architecture sont des artefacts spécifiques qui décrivent le système d'intérêt d'après une perspective donnée. Dans une description d'architecture, un modèle d'architecture peut être englobé dans une ou plusieurs vues. Les modèles d'architecture doivent se conformer à un type de modèle spécifique, qui est une convention pour un type de modélisation. Dans ce contexte, comme un point de vue gouverne une vue, un type de modèle gouverne un modèle, car ils établissent les conventions qui permettent la construction, l'interprétation, et l'utilisation des modèles.

3.3.4 Correspondance architecturale

Les correspondances architecturales peuvent être utilisées afin de définir des relations entre les différents éléments architecturaux (AD Element dans la Figure 3-6 extraite de [2]). Un élément architectural est n'importe quelle construction qui prend part à la description d'architecture. En conséquence, les vues, points de vue, acteurs, préoccupations, types de modèle, modèles peuvent être considérés comme des éléments architecturaux. Nous pouvons voir l'élément architectural comme une classe mère de chaque concept décrit dans le standard ISO/IEC/IEEE 42010.

FIGURE 3-6 CORRESPONDANCE ARCHITECTURALE ET RÈGLE DE CORRESPONDANCE

D'un point de vue abstrait, une correspondance peut être considérée comme le concept *dependency* en UML, car il permet de lier la majorité des éléments du modèle conceptuel : *namedelements* pour les *dependency* UML et *ADElements* pour les *correspondences*. De plus, comme les dépendances en UML peuvent être gouvernées par des contraintes OCL, les correspondances architecturales peuvent être gouvernées par des règles de correspondance. Les règles de correspondance sont utilisées pour renforcer la relation d'une description d'architecture (ou entre différentes descriptions d'architecture). En général, le standard ISO/IEC/IEEE 42010 établit que les correspondances et les règles de correspondance sont utilisées pour exprimer et renforcer les relations architecturales comme la composition, le raffinement, la cohérence, la traçabilité, la dépendance, la contrainte et l'obligation.

3.3.5 Langages de description d'architecture

Selon le standard ISO/IEC/IEEE 42010, un langage de description d'architecture (ou Architecture Description Language, ADL) est n'importe quelle forme d'expression utilisée

dans une description d'architecture. La Figure 3-7 (extraite de [2]) montre comment le concept d'ADL s'articule autour des concepts de l'ontologie définie dans le standard.

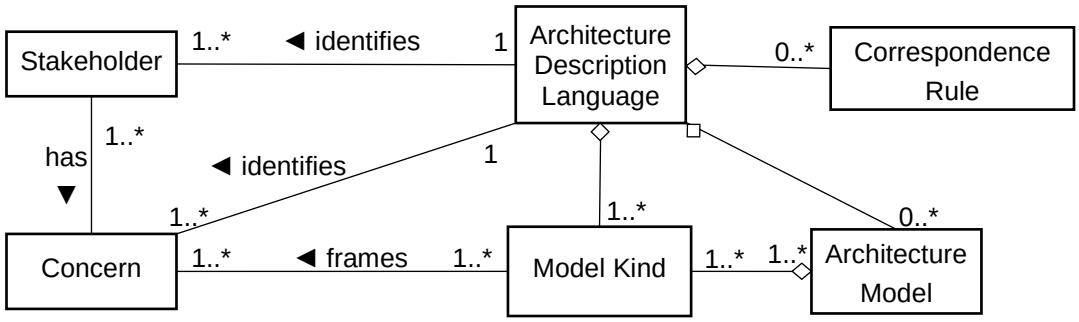


FIGURE 3-7 LANGAGE DE DESCRIPTION D'ARCHITECTURE DANS LE STANDARD 42010

Un ADL fournit un ou plusieurs types de modèle comme un moyen de cadrer des préoccupations pour une audience d'acteurs. Un ADL peut cibler précisément un seul type de modèle, ou plus largement en fournissant plusieurs types de modèle, optionnellement organisés en point de vue. Un ADL est souvent couplé avec un outil automatisant la création, l'utilisation et l'analyse de ses modèles.

Les ADL peuvent être classés en trois catégories [58]:

1. Graphiques informels : Ces ADL ont été pendant longtemps le seul moyen de décrire les architectures. Tout en fournissant une documentation utile, le niveau d'informalité a limité l'utilité des descriptions d'architecture [59], [60].
2. Langage de description d'architecture formel : depuis le début des années 90, la recherche s'est portée sur la définition de plusieurs ADL formels. Le développement de chacun de ces ADL a été dirigé par de légères différences portant sur les éléments architecturaux, sur les différentes syntaxes et sémantiques, ciblant un domaine d'opération très précis, ou seulement applicable à une pratique de travail particulière. Par exemple, des ADL dédiés ont été développés pour les systèmes embarqués temps réel (comme AADL [61], EADL [62], EAST-ADL [63]), l'architecture de ligne de produits (Koala [64]), les systèmes dynamiques (π-ADL [65]). De plus, plusieurs langages d'architecture dédiés à l'analyse ont été proposés pour gérer la disponibilité, la fiabilité, la sécurité, la sureté, la consommation de ressources, la qualité des données et l'analyse des performances (Fractal [66], TADL [67]). Cependant, cet effort remarquable de développement n'a pas réussi à séduire le milieu industriel, pour les raisons analysées dans [68]–[71] : les ADL sont rarement supportés par des outils matures, très peu documentés, concentrés sur des problèmes et des besoins très précis, et ne laissent aucune place pour les étendre avec de nouvelles caractéristiques.
3. UML et notations basées sur UML : UML a été évoqué comme un potentiel successeur aux différent ADL existants, permettant de surmonter certaines de leurs limitations. Plusieurs propositions ont été faites afin d'étendre UML 1.x dans le but

de modéliser les architectures logicielles [72]–[76], puis d'autres pour étendre UML2. Bien qu'UML2 introduise de nouveaux concepts permettant de le rendre plus approprié à la description d'architecture, beaucoup de travaux académiques ont cherchés à l'étendre afin de répondre à des préoccupations spécifiques [77]–[80]. En conséquence, de nombreux profils UML et des extensions ont été proposées pour modéliser différentes préoccupations architecturales, augmentant ainsi encore plus la prolifération des langages de description d'architecture.

3.4 Conclusion

Dans ce chapitre, nous avons présenté différents moyens pouvant être employés aujourd'hui en industrie afin de gérer les spécificités du flot de conception des systèmes sur puce, en vue d'aider à maintenir la cohérence d'un corpus documentaire. De plus nous avons présenté le standard de description d'architecture qui a pour but de faciliter la communication entre différents acteurs impliqué dans la conception d'un système en leur fournissant un moyen de communication commun.

Dans le chapitre précédent, nous avons vu que certains cycles de développement impliquent une cohérence forte. Cependant, cette cohérence est assurée au détriment du temps de développement et d'une inertie face aux changements très importante. D'autres cycles de développement, permettent d'être plus réactif aux changements de spécifications, mais à leur tour nécessitent un sacrifice de la cohérence. Un outillage approprié est donc nécessaire. Bien que ces outils permettent aux équipes géographiquement distribuées d'interagir plus efficacement, aucun moyen présenté dans ce chapitre ne permet de garantir systématiquement la cohérence d'un ensemble de documents impliqué dans le flot de conception d'un système complexe.

En effet, les outils collaboratifs présentés sont agnostiques à la fois de l'architecture et de la description d'architecture des systèmes développés. Ces outils n'étant pas conscient de l'architecture d'un système, ni même des interactions entre les documents, ils ne peuvent pas propager les changements intervenant sur un document. Avoir une description d'architecture cohérente est par conséquent impossible en utilisant uniquement ces outils.

Notre approche, présentée dans le chapitre suivant, propose d'utiliser et d'enrichir certains concepts de la description d'architecture dans le but d'en faciliter la création et le maintien et d'ajouter la notion dynamique absente dans le standard (évolution des documents).

Chapitre 4
Une approche pragmatique des problèmes de cohérence

Sommaire

Comme nous l'avons vu, aucun outil ne permet de gérer la cohérence efficacement en tenant compte des spécificités précédemment identifiées (développement concurrent, agilité, hétérogénéité des formats). De plus, la méconnaissance de la description d'architecture par les acteurs rend impossible une propagation complète et systématique des changements [81]. Notre proposition vise à imposer explicitement un modèle de cohérence au flot de conception des systèmes sur puce grâce à une contribution en deux points :

- Fournir aux acteurs un moyen de capturer et de maintenir **les liens de cohérence inter-documentaire** en contexte de développement itératif et agile.
- Exploiter ces liens pour aider les acteurs à maintenir **systématiquement** la cohérence.

L'approche proposée dans ce manuscrit se base et étend sur le standard de description d'architecture ISO/IEC/IEEE 42010.

Notre proposition vise à permettre la définition explicite de la description d'architecture ainsi que sa maintenance tout au long du développement du système. Pour cela, et au vu du caractère industrielle de cette thèse, notre approche se base et étend le standard de description d'architecture ISO/IEC/IEEE 42010 afin d'y ajouter la notion de dynamicité aujourd'hui absente.

Dans ce chapitre, nous commencerons par mettre en évidence les liens implicites entre les documents impliqués dans le flot de conception des systèmes sur puce et représentant la duplication d'information sémantique. Nous justifierons ensuite de l'utilisation du standard de description d'architecture afin d'expliciter les liens. Dans un deuxième temps, nous présenterons une extrapolation du concept de transformation de modèle, issu du domaine de l'IDM. Cette extrapolation permet d'identifier les informations nécessaires pour définir complètement les liens de cohérence. Nous présenterons ensuite l'approche proposée, et plus spécifiquement les deux parties de l'approche, à savoir la définition des liens et l'exploitation de ces liens pour maintenir la cohérence. Enfin, nous détaillerons les extensions du standard que nous proposons afin d'enrichir le standard 42010 avec les informations nécessaires à notre approche.

4.1 Duplication des informations dans le flot de conception

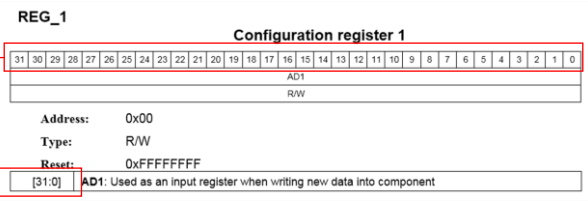
Les documents créés, lors du développement d'un système sur puce, sont souvent le résultat du raffinement, de la composition ou de la modification de documents préexistants. En effet, la production des nouveaux documents est due à un besoin de formaliser une idée sur le système en développement et cette idée est le résultat de volontés ou de préoccupations suivants les exigences du client. En conséquence, nous avons une **duplication des informations** entre les documents nouvellement créés et les documents utilisés préexistants.

En observant la Table 4-1 représentant un registre décrit dans plusieurs formats et à différents moments du flot de conception des systèmes sur puce, nous pouvons nous convaincre de l'existence de cette duplication d'information. En effet, la modification de la taille du registre dans la description du registre VHDL, par exemple, nécessitera la modification de tous les autres documents afin de maintenir cet ensemble de documents cohérent. Nous pouvons donc faire l'hypothèse qu'il existe entre tous ces documents des liens représentant la duplication d'information sémantique (lien dans la table). Cependant,

ces liens sont aujourd'hui uniquement définis dans l'esprit des acteurs du flot. Seuls les acteurs ayant créés les documents savent dans quels documents préexistants les informations ont été extraites. Nous avons donc des liens de cohérence **implicites et informels**. Afin de maintenir la cohérence, et surtout rendre les acteurs conscients de l'impact de leurs modifications, l'expression de ces liens doit être explicite.

TABLE 4-1 LIEN DE COHÉRENCE INTER-DOCUMENTAIRE

Spécification TEXTE


REG_1

Configuration register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD1																															
R/W																															

Address: 0x00
Type: R/W
Reset: 0xFFFFFFFF

[31:0]	AD1: Used as an input register when writing new data into component

Description du registre IP-XACT

```
<spirit:register>
  <spirit:name>REG_1</spirit:name>
  <spirit:description>Configuration register 1</spirit:description>
  <spirit:addressOffset>0x00</spirit:addressOffset>
  <spirit:size>32</spirit:size>
  <spirit:access>read-write</spirit:access>
  <spirit:reset>
    <spirit:value>0xFFFFFFFF</spirit:value>
  </spirit:reset>
  <spirit:field>
    <spirit:name>AD1</spirit:name>
    <spirit:description>Used as an input register when writing new data into component</spirit:description>
    <spirit:bitOffset>0</spirit:bitOffset>
    <spirit:bitWidth>32</spirit:bitWidth>
    <spirit:access>read-write</spirit:access>
  </spirit:field>
</spirit:register>
```

Description du registre VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity testReg is

    port (
        REG_1: in std_logic_vector (31 downto 0);
    );
end testReg;
```

Définition de macro C

```
/*!
* \brief
* Register : REG_1
* Configuration register 1
*/

#define UART_DMAC_UART1_data_SIZE                    (32)
#define UART_DMAC_UART1_data_OFFSET                  (0x00)
#define UART_DMAC_UART1_data_RESET_VALUE             (0xFFFFFFFF)
#define UART_DMAC_UART1_data_BITFIELD_MASK           (0x00000000)
#define UART_DMAC_UART1_data_RWMASK                  (0xFFFFFFFF)
#define UART_DMAC_UART1_data_ROMASK                  (0x00000000)
#define UART_DMAC_UART1_data_WOMASK                  (0x00000000)
#define UART_DMAC_UART1_data_UNUSED_MASK             (0xFFFFFFFF)
```

Initialisation de la banque de registres C++

```
#include "base_COMPONENT.hpp"

//----------------------------------------
// Register banks initialization
//----------------------------------------
void base_COMPONENT::init_regs_COMPONENT() {
    std::stringstream ss;
    ss << "REG_1              0x0      32b  0x00  [RW]; /* Configuration register 1 */  \n";
    regs_COMPONENT.set_register_mapping(ss);
}
```

Documentation PDF

4.4.1 Configuration register 1 (REG_1)

Address offset: 0x00

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AD1[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **Configuration register bits**
Used as an input register when writing new data into component

4.2 Transformation de modèles appliquée au flot de conception des systèmes sur puce

Nous souhaitons donner aux différents acteurs impliqués dans le développement d'un système un moyen de formaliser la duplication d'information qui existe entre plusieurs documents. En effet, comme nous l'avons vu, la création de nouveaux documents par les acteurs est souvent faite en extrayant ou en référençant des informations de documents préexistants. Ce fait mène à avoir des fragments d'informations sémantiques redondants et distribués dans plusieurs documents.

La définition de la transformation de modèles [82], [83] issue des pratiques de l'IDM [84], [85] établit qu'une transformation de modèles est un moyen d'assurer qu'une famille de modèles est cohérente, en **générant** des modèles **cibles** à partir de modèles **sources**.

Chaque création de document peut être vue comme une transformation d'un ou plusieurs documents vers un ou plusieurs autres documents. En effet, des documents sources sont utilisés (en extrayant des informations de leurs contenus) pour créer des documents cibles. De plus, la modification d'un document peut également être vue comme une transformation : le document cible (le document après modification) a été généré ou créé à partir d'un document source (le document avant la modification).

Voir la création de nouveau document ou la modification de document existants comme une transformation entraine la comparaison entre les organes de computations effectuant les transformations de modèles (programmes, script etc..) et les différents acteurs d'un flot de conception effectuant les modifications ou création de document.

Il apparait alors clairement que définir formellement et complètement les liens de cohérence inter-documentaire revient à définir les transformations des documents sources vers les documents cibles. De plus, notre approche vise à finaliser cette extrapolation, en assurant un modèle de cohérence explicite dans le flot de conception des systèmes sur puce.

4.3 Expliciter les transformations entre les documents

La première partie de l'approche que nous proposons consiste en la définition complète des liens de cohérence, basé sur des études faites sur les transformations [86]–[90]. De plus, nous explicitons les meilleures pratiques, selon nous, permettant de faciliter à la fois l'adoption de l'approche, en insérant la création des liens dans les pratiques de travail, mais également de la maintenance des liens au cours du développement.

4.3.1 Définition des transformations

Pour commencer, il est important de noter qu'à la différence des transformations de modèles de l'IDM, les documents cibles ne sont pas forcément générés au moyen de programmes informatiques. En effet, le développement d'un système est par essence une

activité humaine, la création de nouveaux documents nécessitera donc l'ajout de nouvelles informations par les acteurs du flot. Cependant, ces documents cibles non-générés automatiquement sont tout de même issus de transformations. L'organe de computation est, dans ce cas, simplement l'esprit humain. Les acteurs du système extraient eux même l'information sémantique utile des documents sources pour la retranscrire et ajouter de nouvelles informations dans les documents cibles. Il apparait alors, qu'un des critères fondamentaux de la définition des transformations est tout d'abord son niveau d'automatisation. Le niveau d'automatisation a donc deux valeurs possibles : manuel (la génération des documents cibles est effectuée par des acteurs humains) ou automatique (la génération est effectuée par des programmes ou scripts).

Chaque transformation appliquée à un document a un temps d'exécution. Nous pouvons déduire de cette caractéristique le temps moyen d'une révision sur un document afin d'estimer le temps nécessaire pour assurer à nouveau la cohérence, en cas de modification d'un document source.

Une transformation est également définie par ses sources et ses cibles. Il convient donc d'enregistrer le plus de méta-information possible sur ces documents : nombre de documents, formats, structures. De plus, cette définition des sources et des cibles doit faire apparaitre le lien qui relie les acteurs impliqués dans le développement du système aux documents.

4.3.2 Approche basée sur ISO/IEC/IEEE 42010

Le standard ISO/IEC/IEEE 42010 offre une base permettant de construire une description d'architecture en l'adaptant au contexte de développement des systèmes sur puce. En effet, les concepts de *view* et de *viewpoint* mettent en évidence la réalité de l'industrie : chaque **acteur** impliqué dans le développement d'un système a une **vue** (gouvernée par un **point de vue**) sur le système développé. Cette vue est décrite grâce à la combinaison de plusieurs *architecture model* (représentant des informations) qui permettent à l'acteur d'exprimer son idée sur le système. Ces *architecture model* peuvent être partagés par plusieurs vues représentant ainsi la **duplication d'informations** entre les équipes. Le concept sous-jacent est que chaque document est créé en extrayant des informations d'un ou plusieurs documents préexistants, mettant en lumière la duplication d'information sémantique dans plusieurs documents.

De plus, ce standard confirme le problème de gestion de la cohérence identifié dans STMicroelectronics. En effet, il établit clairement qu'avoir une description d'architecture entièrement cohérente est parfois un objectif inatteignable pour des raisons de temps, d'effort ou d'informations insuffisantes. Cependant, le standard encourage à enregistrer toutes les incohérences connues en utilisant le concept de *correspondence*.

La définition de lien que nous proposons est basée sur le concept de correspondance introduit dans le standard ISO/IEC/IEEE 42010 (voir Figure 3-6). Les correspondances, telles que définies dans le standard, permettent d'exprimer des relations entre différents éléments architecturaux. Notre proposition vise donc à étendre cette définition générale des correspondances afin de faire apparaitre clairement les propriétés caractérisant les transformations.

De plus, notre approche vise à pallier deux problèmes majeurs du standard de description d'architecture : la représentation d'information dynamique et le maintien de la description d'architecture tout au long du développement d'un système.

Le standard de description d'architecture ISO/IEC/IEEE 42010 laisse de côté toute la représentation dynamique d'une description d'architecture. En effet, les notions de flot de conception ou d'évolution des documents décrivant le système ne sont pas abordées dans le standard. Notre proposition permet de mettre en lumière ces notions et d'apporter, grâce à la définition de *correspondence* que nous enrichissons, la description du dynamisme des documents et des pratiques mises en œuvre pour développer un système.

Enfin, décrire correctement une description d'architecture prend énormément de temps. Chaque interaction, chaque équipe (voire individu), chaque document doit être capturé. Ce travail doit être effectué tout au long du développement du système, sous peine de décrire des pratiques qui n'ont plus cours ou plus de sens. Cette mise à jour fréquente doit être effectuée par les développeurs des différentes équipes qui risquent de voir en cette tâche supplémentaire un travail négligeable, comme la documentation ou les tests dans une certaine mesure. Notre approche permet d'automatiser la création et le maintien de la description d'architecture, et donc de sa cohérence, tout au long du flot de conception.

4.3.3 Création des liens dans les pratiques de travail classique

La définition des liens entre les documents ne peut pas être faite en ajoutant une étape dans le flot de conception avec un outil dissocié. Adopter cette approche nécessiterait un investissement temporel et personnel de la part des acteurs beaucoup trop important, qui verrait ceci comme une perte de temps. L'ensemble des liens serait rarement mis à jour, comme les documentations ou les tests par exemple. Dejours [91] affirme qu'un nouveau comportement ne peut pas être appliqué instantanément. L'installation de nouveaux comportements, ou de nouvelles méthodes de travail, implique forcément une résistance aux changements de la part des acteurs, obligeant à combattre les anciennes méthodes de travail ou les anciens comportements. Pour éviter cette résistance aux changements, la solution la plus simple et la plus largement adoptée est d'éviter les changements brutaux afin de permettre une adaptation progressive. De plus, même avec la meilleure volonté possible, les acteurs du flot seraient susceptibles d'oublier de créer les liens.

Dans cette optique, l'étape de définition des liens devra être totalement intégrée dans les méthodes de travail usuelles. Lorsqu'un acteur termine la création ou la modification d'un document, il publie le document nouvellement créé dans un *dépôt* (en utilisant un outil de gestion de versions). Cette étape habituelle fournit un point d'entrée à la création des liens. En effet, nous pouvons profiter de la publication des documents créés pour extraire et dérouter des méta-informations (définies précédemment) sur les documents. L'acteur publiant les documents peut également être sollicité afin de fournir des informations supplémentaires sur la transformation effectuée afin de créer ou modifier les nouveaux documents. Cette approche permet de définir les liens au plus proche du *dépôt*.

4.3.4 Rendre les systèmes de gestion de version conscients de l'architecture des systèmes sur puce

Les *dépôts* actuels ne permettent pas de maintenir la cohérence d'un corpus documentaire. En effet, ces dépôts de versions permettent de gérer efficacement les différences entre plusieurs versions d'un même document lorsque ces documents sont assez simples (texte, code…). Dès lors que les documents versionnés sont des documents complexes (modèles, images, exécutables), ces outils ne sont plus capables de gérer efficacement les différences à un grain fin.

De plus, comme nous l'avons vu dans le chapitre précédent, les outils de gestion de versions ne peuvent pas, dans le cas présent, aider à propager les modifications appliquées à une version de documents. En effet, ces outils n'étant pas conscient de l'architecture d'un système, ni même des interactions entre les documents, ils ne peuvent pas propager les changements intervenant sur un document. L'approche vise à pallier cette méconnaissance de l'architecture de la part des dépôts de version en les rendant conscients de la description d'architecture, à la façon d'un plugin, afin d'aider les acteurs du flot à propager systématiquement les modifications.

4.3.5 Granularité des fragments

La granularité d'un fragment représente l'élément le plus fin liable. Une granularité microscopique (ou grain fin) des liens a pour avantage de réduire le dérangement des utilisateurs par des fausses alertes. En effet, lorsque les liens ciblent précisément des fragments de documents, si un fragment de ces documents est modifié, nous pouvons alors informer précisément quelle partie de document a été modifiée et en conséquence quel fragment du document impacté doit être à son tour être modifié afin de rester cohérent. Des liens microscopiques seront donc possiblement plus précis et plus efficace dans la gestion du changement. Cependant, avoir des éléments liables fins oblige à être capable d'extraire et d'analyser le contenu de tous les formats de documents. Cependant, pour les objets complexes (comme les fichiers binaires ou les documents non structurés par exemple) l'accès à ce niveau de granularité peut être impossible à mettre en œuvre.

De plus, une granularité fine des liens aura pour effet de rendre la création et la maintenance des liens beaucoup trop contraignante pour l'utilisateur. Par exemple, certaines documentations ou spécifications écrites en langage naturel peuvent atteindre des centaines de pages. Choisir un grain fin (niveau titre ou sous-titre par exemple) pour ce genre de document volumineux aura pour effet de devoir lier plus d'une centaine d'éléments. De plus, chaque information sémantique disséminée dans ces fragments peut être partagée par plusieurs documents, soit d'autant plus de liens à créer. La maintenance de ces liens serait une tâche beaucoup trop longue et complexe pour être réalisée par un acteur.

D'un autre côté, des liens à granularité macroscopique (ou gros grain) peuvent facilement noyer l'acteur dans un océan de fausses alertes. En effet, si le grain est trop gros, une modification d'une seule information sémantique sera perçue comme une modification de la combinaison d'informations sémantiques contenues dans le grain. L'acteur risque alors d'être informé d'une modification qui ne l'impacte pas directement. Ces informations parasites peuvent amener l'acteur à négliger l'intégralité des notifications, rendant le système totalement inefficace.

En revanche, créer des liens avec une granularité macroscopique permet d'être indépendant du contenu des documents. En effet, si nous considérons que le grain le plus fin que nous pouvons lier est le niveau d'un document dans sa globalité, il ne sera pas nécessaire de connaitre son contenu pour propager une modification. De plus, plus le grain est macroscopique, plus la maintenance et la création des liens seront facilitées : le nombre de liens à gérer sera forcément plus faible.

La taille des fragments d'informations sémantiques liables (macroscopique ou microscopique) a un impact direct tant sur l'efficacité que sur l'ergonomie des liens, chacun ayant des avantages et des inconvénients. Il convient alors de trouver le juste milieu. Nous avons choisi de fournir par défaut un grain macroscopique : l'élément le plus fin étant le niveau document. Nous avons, avant tout, cherché à avoir un élément facilement manipulable par les acteurs et ne demandant pas un investissement de temps excessif. Bien que nous prenions le risque avec cette granularité de déranger les utilisateurs avec des notifications de changements qui ne les impactent pas. Afin de justifier notre choix de granularité, nous prenons en exemple la documentation technique du STM32F427 [92]. Ce document est composé de 239 pages, si nous choisissions de fragmenter ce document au niveau titre, nous obtiendrions 118 éléments. À présent, si nous nous référons à la Table 4-1 et que nous supposons que chaque fragment liable peut-être répliqué dans cinq documents différents, nous obtenons un total de 590 liens à créer et à maintenir pendant le développement du système et ceci pour un seul document. Cet exemple démontre que l'utilisation d'un grain fin n'est pas toujours adaptée à la maintenance de cohérence.

De plus, de nombreux outils de différenciation existent [93]–[95], suffisamment efficaces pour comparer deux documents (dans le cas qui nous intéresse, deux versions d'un même

document) et permettre ainsi de ne montrer que ce qui a réellement changé. Cette utilisation des outils de différenciation permet de ne pas passer trop de temps à analyser l'impact d'une modification. Néanmoins, l'utilisation de ces outils reste limitée à certains formats spécifiques.

Cependant, notre approche laisse les utilisateurs libres d'adopter leurs propres granularités de fragments. En effet, les acteurs peuvent, sur la base du standard, définir la structure des documents (en modulant la représentation des *ADElements* relatifs à une correspondance) participant à la description de l'architecture et ainsi donner la possibilité de lier des éléments possiblement très fins. En revanche, ils doivent être conscients que vouloir définir un grain plus fin sera plus compliqué à maintenir, avec le risque d'avoir une explosion du nombre de liens [96]–[98].

4.4 Exploitation des liens pour propager les modifications

Les liens créés précédemment permettent d'aider les utilisateurs à propager les modifications apportées aux documents. En effet, lors de la publication d'une modification d'un document, comme lors de la création des liens, les méta-informations sur les documents publiés sont extraites. Les méta-informations caractérisent les transformations et contiennent : la nature de la transformation (manuelle ou automatique), le nom de l'acteur responsable de la transformation, les documents impliqués, la localisation des documents, le numéro de transaction du gestionnaire de versions, le commentaire de publication et un horodatage de la transformation. La description d'architecture enregistrée (somme des transformations et des méta-informations sur les documents) peut être analysée afin de vérifier si les documents modifiés sont des sources ou des cibles de transformations. Si tel est le cas, une notification est envoyée aux acteurs responsables de documents partageant de l'information pour les prévenir d'une modification. Cette notification est le cœur de notre approche quant à la propagation des changements. En effet, celle-ci contient toute les informations dont nous disposons (documents modifiés, date, acteur ayant effectué la modification…). Ces informations, complémentaires au message de publication (*commit message)*, et caractéristique d'une transformation, permettent à l'acteur destinataire de prendre immédiatement conscience de la possibilité d'une incohérence entre les documents. La Figure 4-1 présente un résumé de ces méta-informations sous forme d'un *Feature Model*.

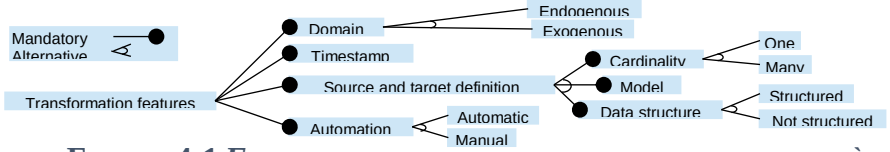


FIGURE 4-1 *FEATURE MODEL* DE TRANSFORMATION DE MODÈLES

4.4.1 Adaptation du système en réponse aux actions des utilisateurs

L'approche proposée vise à rendre cette maintenance automatisée et obligatoire, en adaptant la description de l'architecture par rapport aux décisions prises par les acteurs suite à des modifications de documents. En effet lorsqu'une modification d'un document est détectée, les acteurs sont informés de cette modification. À ce stade, trois choix peuvent s'offrir à eux :

- **Non impacté** : L'acteur peut choisir de signaler qu'il est au courant de la modification d'un document partageant de l'information avec un des documents dont il est l'auteur, mais que cette modification ne l'impacte pas. Le fragment d'information sémantique modifié n'est pas partagé par son document. Dans ce cas, les deux documents sont toujours cohérents. La Figure 4-2 illustre ce choix. Nous avons deux représentations d'un même registre (une spécification au format MS Word et une description du registre au format VHDL). Ces deux documents sont liés par un lien de cohérence. Une modification intervient sur la spécification (passage de Version 1 à Version 2) : la description du registre passe de « Configuration register 1 » à « Configuration register 2 ». Cette modification n'impacte pas la description du registre au format VHDL, ce n'est pas cette information sémantique qui était visée par le lien de cohérence.

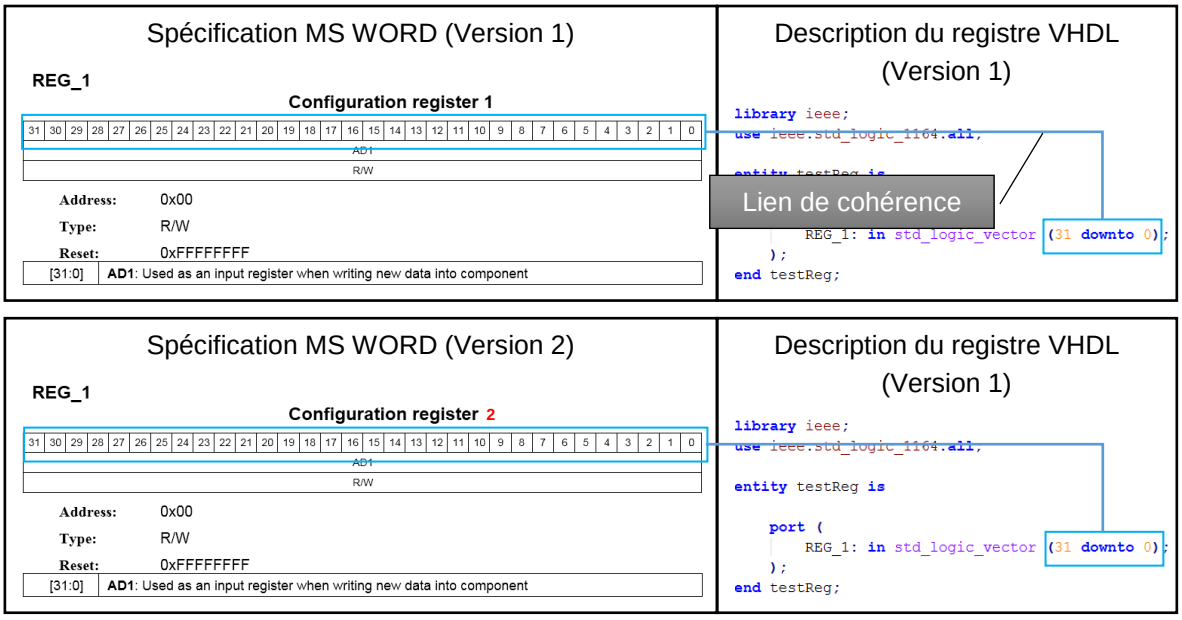


FIGURE 4-2 EXEMPLE D'UNE MODIFICATION NE GÉNÉRANT PAS D'INCOHÉRENCES

- **Mise à jour** : La deuxième option est une nécessité pour l'acteur de modifier son document afin de maintenir la cohérence entre le document modifié et son document. L'information sémantique modifiée dans le document est partagée par le document dont l'acteur est propriétaire. Une fois la mise à jour du document faite par l'acteur, les documents sont à nouveau cohérents. La Figure 4-3 illustre ce choix. Nous avons deux représentations d'un même registre (une description d'un registre en IPXACT et une initialisation de la banque de registres en C++). Ces deux documents sont liés par un lien de cohérence. Une modification intervient sur la description en IPXACT (passage de Version 1 à Version 2) : l'accès du registre passe de lecture/écriture à lecture uniquement. Cette modification impacte la banque de registre C++ qui passe à la version 2 à son tour pour rester cohérente.

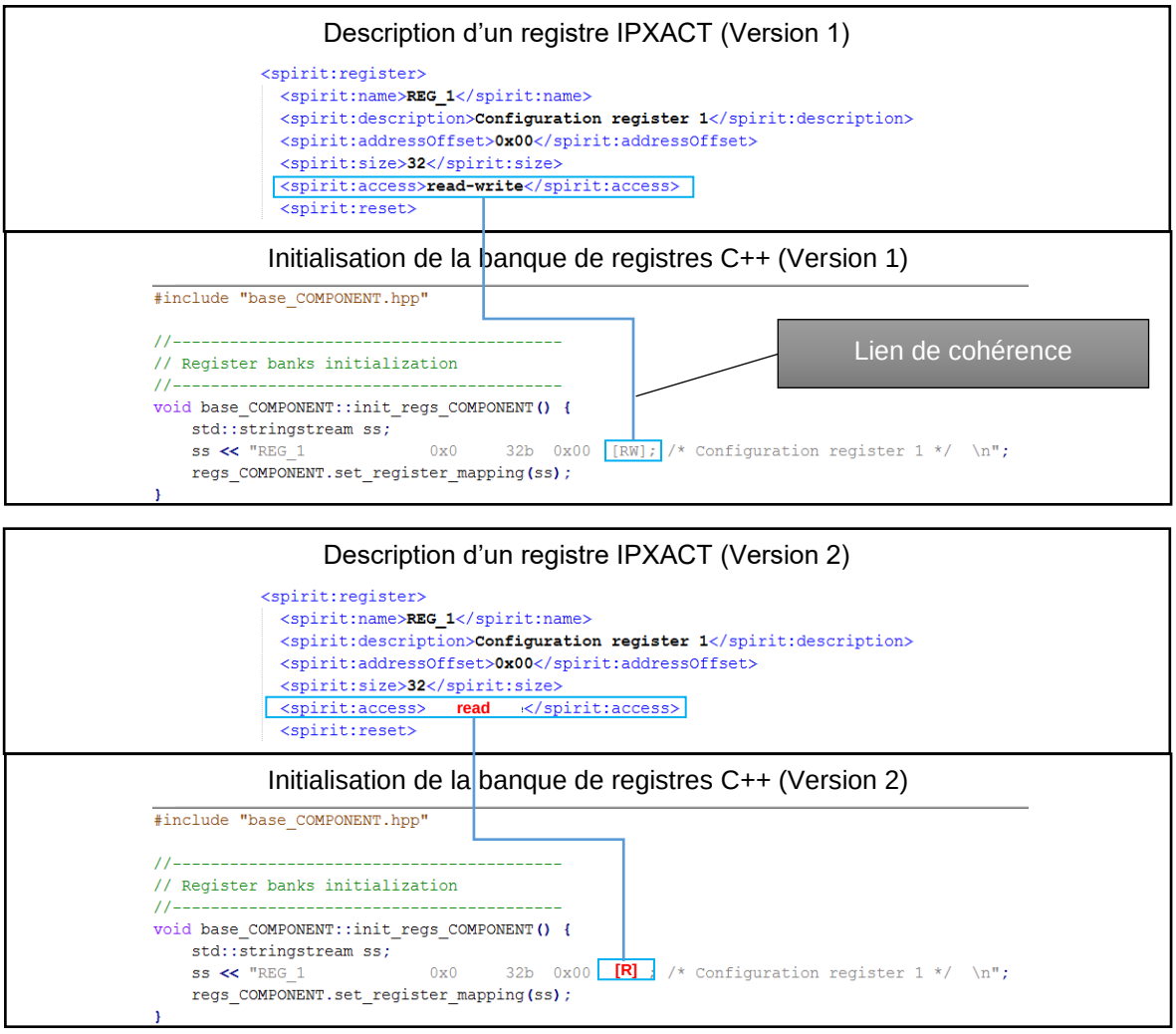


FIGURE 4-3 EXEMPLE D'UNE MODIFICATION NÉCESSITANT UNE MISE À JOUR

- **Dégradation** : La dernière option qu'a l'acteur après la modification d'un document partageant de l'information avec un document dont il est le propriétaire est la dégradation du corpus documentaire. Les acteurs peuvent choisir de volontairement casser la cohérence du corpus documentaire en dégradant un lien. Avec cette option l'acteur exprime le fait qu'il a bien pris connaissance de la modification. Son document devrait être modifié pour rester cohérent, mais qu'il refuse d'appliquer cette modification à son document. Les acteurs d'un flot de conception n'ont pas toujours le temps de propager les modifications, ou bien ils peuvent être en désaccord avec la modification de l'information sémantique ayant eu lieu. Dans ce cas, le lien entre les documents est brisé. Cependant, comme l'encourage le standard ISO/IEC/IEEE 42010, l'incohérence est enregistrée. La Figure 4-4 illustre ce choix. Nous reprenons les deux documents de l'exemple précédent. Cette fois-ci, après la modification de la description du registre IPXACT, l'acteur responsable de la banque de registres C++ décide de dégrader la cohérence. Dans ce cas les deux documents ne partagent plus cette information sémantique, le lien est donc supprimé.

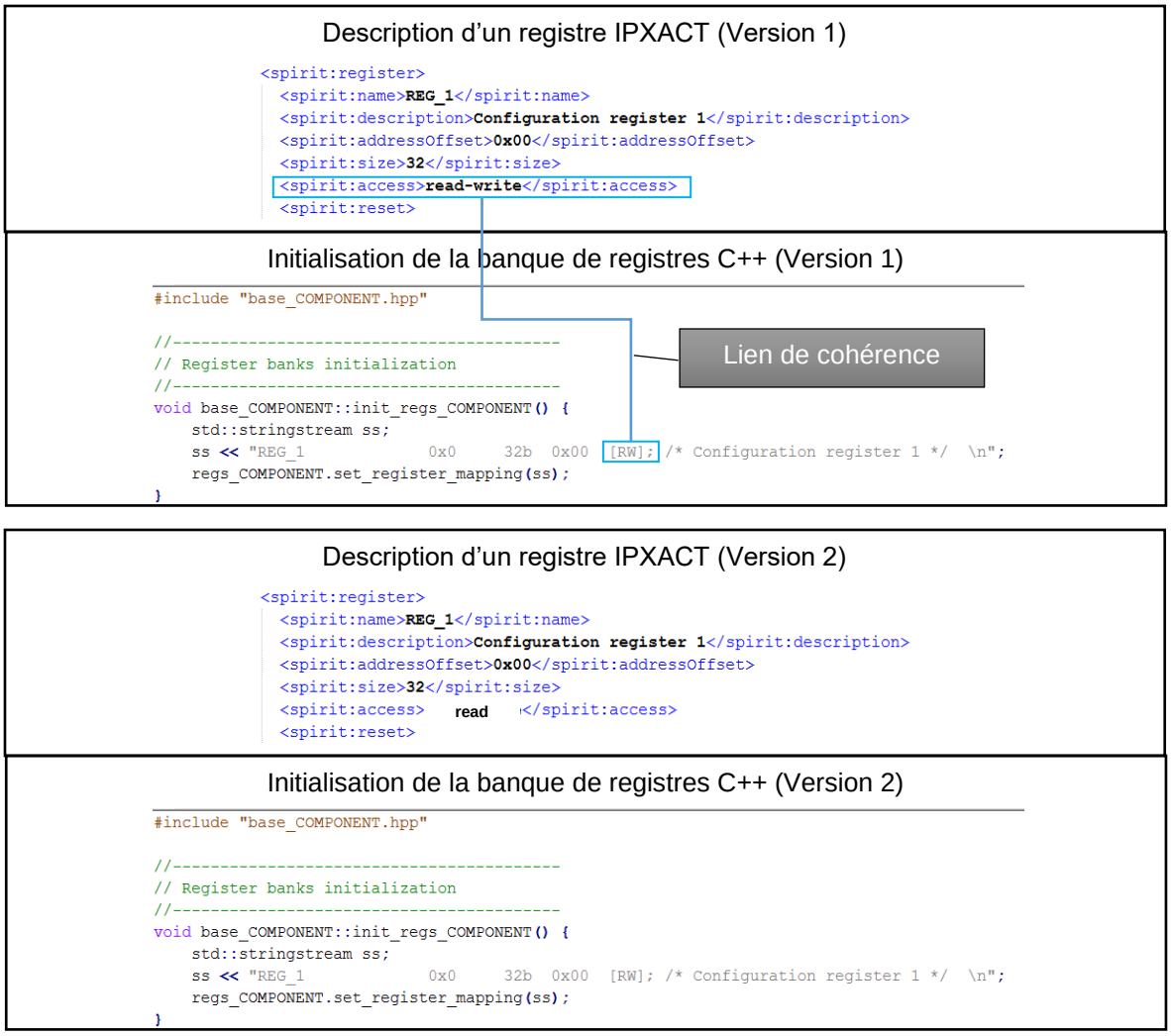


FIGURE 4-4 EXEMPLE D'UNE DÉGRADATION DE LA COHÉRENCE

4.4.2 Émergence de processus

La définition des liens représentant le partage d'information entre les documents permet la création d'une cartographie de tous les documents décrivant le système. En effet, ces correspondances mettent en évidence la façon dont les différentes équipes interagissent entre elles. De plus, l'approche permet de visualiser le rayon d'action de chaque acteur en les associant à un ensemble de document. Chaque acteur du système est associé aux différents documents qu'il a créés ou modifiés. En utilisant cette méthode, le processus réel suivi par les acteurs du système émerge de façon pragmatique à partir de l'observation des pratiques de travail.

La maintenance d'une description d'architecture, si celle-ci est définie, est rarement effectuée au cours du cycle de développement du système pour plusieurs raisons de temps, d'investissement, de mises à jour trop fréquentes nécessaires. Nous donnons en Annexe 1 la modélisation du flot de conception des systèmes sur puce suivi dans STMicroelectronics. Le flot de conception a été modélisé en travail préparatoire de cette de thèse grâce au langage BPMN [99], [100] pour ces qualités de représentation graphique de processus. Malgré le temps passé en réunion pour obtenir les informations nécessaires, ce processus n'est pas complet, et trop peu précis pour être exploité dans STMicroelectronics.

Habituellement, l'utilisation d'un processus dans les pratiques de travail implique de créer le processus en amont de la phase de développement et de forcer les acteurs à travailler suivant les étapes qui sont définis. Dans les faits, si les processus définis en amont ne sont pas suivis scrupuleusement, l'approche n'est plus réellement efficace [101]–[103]. En effet, les documents sont souvent modifiés, les acteurs sont parfois obligés de mettre à jour les documents, fixer des problèmes, sans que cela n'apparaisse dans le processus préétabli. Les pratiques de travail effectuées par les acteurs divergent de plus en plus du processus défini en amont. Ce processus est alors constamment obsolète.

L'approche proposée nous permet de générer le processus à chaque instant, en prenant en compte toute les actions effectuées par les acteurs. En effet, la description d'architecture créée est représentée par la somme des documents et des actions (concrétisation des liens), soit le processus suivi réellement par les acteurs. Ce processus généré par l'approche a alors la caractéristique d'être en tout temps mis à jour et toujours aligné avec les pratiques de travail effectuées par les acteurs du système.

La possibilité d'avoir une visualisation du fonctionnement réel de la conception des SoC au travers d'un processus mis à jour en permanence est très utile. En effet, les cadres peuvent s'en servir afin de savoir quelles étapes ou quelles activités ont besoin d'un investissement plus important ou pour recadrer les actions des acteurs. De plus, les différents acteurs ont, avec ce processus, la possibilité de connaitre l'impact d'une modification de document qu'ils publient. En outre, les acteurs ont la possibilité de suivre la propagation de leurs

modifications au fil des approbations et modifications effectuées par les autres acteurs sur les documents impactés.

4.5 Modèle de cohérence induit par l'approche

L'application de l'approche et de la méthodologie proposée impose, par essence, un modèle de cohérence moins faible que le modèle *eventual consistency* appliqué implicitement au flot de conception. Le modèle imposé se situe à la frontière entre le modèle *Eager Release consistency* (ERC) et le modèle *Write-follows-Read consistency* (WFRC).

D'une part, l'application du modèle ERC au flot de conception des systèmes sur puce est marquée par la propagation des modifications uniquement vers les documents contenant la même information sémantique. L'identification des liens de cohérence inter-documentaire et la propagation des modifications proposées dans l'approche permettent de réaliser cette propagation ciblée. Cependant, le modèle ERC est tout de même assez rigide et impose l'utilisation d'un système de synchronisation sacrifiant la disponibilité des documents pendant la propagation d'une modification. Ce sacrifice ne peut pas être toléré dans notre contexte, le retard engendré par ces blocages est incompatible avec les contraintes du marché (TTM …).

D'autre part, l'application du modèle WFRC au flot de conception est marquée par la disponibilité des documents à chaque instant, par la propagation des modifications en scrutant les modifications de documents survenant après la lecture de la modification initiale. Cependant, ce mode de propagation laisse les incohérences exister tant qu'un acteur n'a pas lu la modification et propagé celle-ci. L'approche que nous proposons applique ce modèle de cohérence. En effet, la disponibilité des documents étant un critère fondamental dans notre contexte, nous laissons les incohérences exister dans le corpus documentaire, tout en les référençant dans la description d'architecture. De plus, la propagation des modifications intervient lorsqu'un acteur (notifié de la modification) prend connaissance d'une modification et modifie son document impacté en conséquence.

4.6 Adaptation du standard de description d’architecture

Le standard ISO/IEC/IEEE 42010 fournit les concepts clés pour construire une description d’architecture. Cependant, la notion de cohérence dans le standard n’est pas explicitée. Nous proposons donc ici des extensions du méta-modèle défini dans ce standard afin de l’enrichir avec les informations nécessaires à notre approche. Ces extensions sont séparées en trois paquetages : **document**, **correspondance et propagation**.

4.6.1 Représentation des documents

Le méta-modèle du paquetage **document** (Figure 4-5) impose les contraintes quant à la création et l'utilisation des documents dans la description d'architecture.

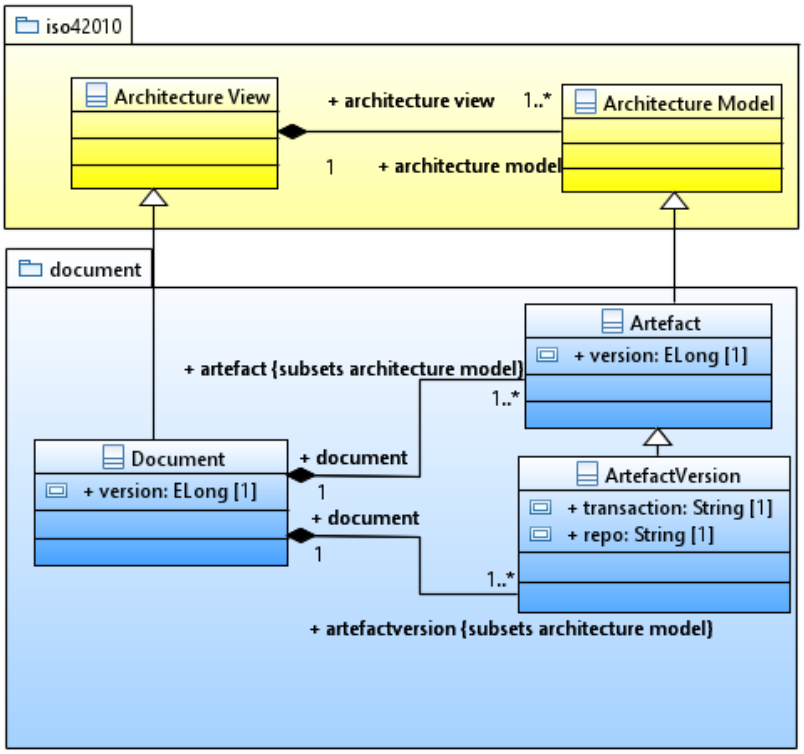


FIGURE 4-5 PAQUETAGE "DOCUMENT"

4.6.1.1 Artefact

L'*artefact* représente n'importe quel document, sans égard pour le format, le contenu (texte, image, données binaires…) ou la structure. L'*artefact* est une extension du concept d'*architecture model* introduit dans le standard ISO/IEC/IEEE 42010. Les *architecture models* sont des *artefacts* spécifiques qui décrivent le système d'intérêt d'après une perspective donnée. Si l'on voit un document comme une séquence d'opérations [51], l'*artefact* permet de représenter le document final, la version courante du document.

4.6.1.2 Artefact Version

L'artefact version représente une version spécifique d'un document. Il contient deux attributs : une chaine de caractère **repo** correspondant à l'URL du dépôt où est stockée la version de l'artefact, et une chaine de caractère **transaction** identifiant le numéro de transaction du dépôt ayant créé cette version du document.

Chaque fois qu'un document est modifié, un *artefact version* est créé et ajouté à la *vue d'artefact*. Si l'on voit un document comme une séquence d'opérations, chaque *artefact version* représente une des opérations. L'ensemble des éléments *artefact version* représente

l'historique des transformations appliquées au document, permettant d'arriver au document final.

4.6.1.3 Document

Un *Document* redéfinit la notion de vue architecturale (ou *Architecture View*). Une description d'architecture selon le standard ISO/IEC/IEEE 42010 est un assemblage de *vues architecturales*, chacune d'elle se focalisant sur un aspect ou une préoccupation du système. Dans notre cas, une description d'architecture est l'ensemble des documents produits par les acteurs. L'extension de la *vue* va donc faire apparaitre cet aspect. Le *document* représente un aspect de la définition d'architecture du système développé en décrivant l'ensemble d'un document. Un document est composé d'un *artefact* et d'autant de *versions d'artefact* qu'il y a de versions du document.

4.6.1.4 Personnalisation des artefacts

Les concepts proposés ici peuvent également être étendus pour cibler les problèmes spécifiques à chaque acteur. Par exemple, des extensions des éléments du paquetage **document** permettent également à l'utilisateur de moduler la granularité des fragments d'information sémantiques qu'il souhaite traiter et lier. La Figure 4-6 montre un exemple de personnalisation des artefacts afin de moduler la granularité des fragments d'information sémantique. Dans ce cas, l'élément *Doc Specific* étend la notion de *document*, il représente l'intégralité d'un document (de format spécifique). L'élément *Doc Specific Fragment* quant à lui étend la notion d'*artefact*. Chaque fragment du document pourra être vu comme un *artefact* à part entière et versionné au travers des *versions d'artefacts*. La même mécanique peut être utilisée pour grouper plusieurs fichiers, si l'utilisateur souhaite traiter des grains plus gros.

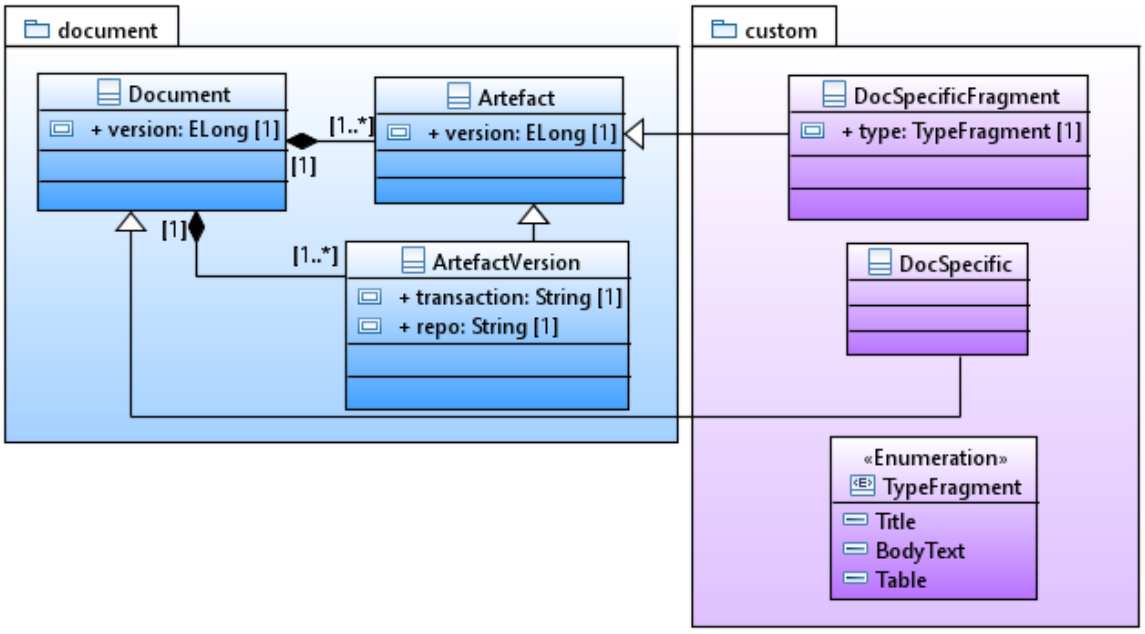


FIGURE 4-6 PERSONNALISATION DES ARTEFACTS

4.6.2 Représentation des correspondances

Le méta-modèle de **correspondance** (Figure 4-7) régit la création et l'utilisation des correspondances dans le cadre de l'approche. Deux types de correspondances ont été définis : *Ownership Correspondence* (OC) et *Transformation Correspondence* (TC).

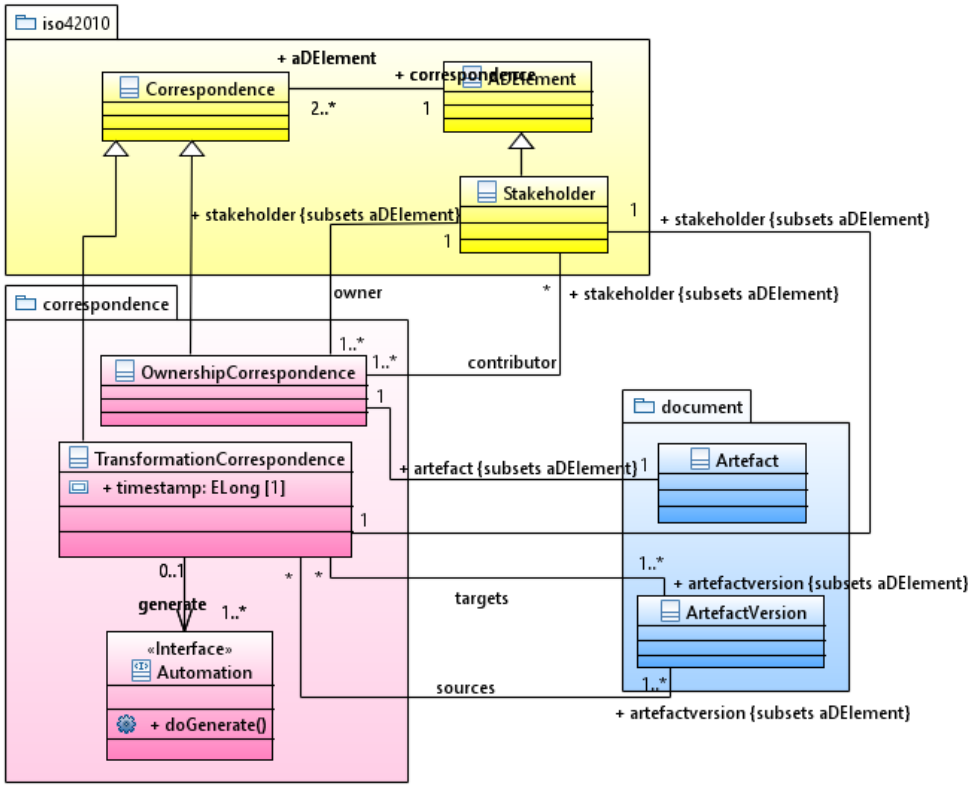


FIGURE 4-7 PAQUETAGE "CORRESPONDENCE"

4.6.2.1 Ownership Correspondence

L'OC lie un document (représenté par un *artefact*) aux acteurs impliqués dans sa création ou son développement. Nous avons choisi l'approche présentée dans [104], le **propriétaire** d'un document est le créateur original du document. Cependant, l'évolution ou la maintenance d'un document n'est pas nécessairement effectuée par le **propriétaire**. Ces acteurs, impliqués dans le développement du document, sont désignés comme des **contributeurs**. Cette distinction permet au **propriétaire** d'avoir tout le temps la priorité pour modifier et réaligner le document. Cependant, le propriétaire d'un document doit pouvoir être modifié au cours du cycle de vie du document. Par exemple, si un acteur du flot vient à changer de poste, il ne pourra plus être responsable des documents.

4.6.2.2 Transformation Correspondence

L'élément TC lie un ou plusieurs documents représentés par des *versions d'artefact* et identifiés comme des **sources** à un ou plusieurs documents identifiés comme des **cibles**. La TC identifie également un acteur comme le responsable de la transformation.

Une TC fonctionne comme un modèle publieur-souscripteur [105], [106]. Un acteur définissant une correspondance, il réfère également les documents utilisés pour créer le nouveau document en définissant les sources et cibles de la correspondance. Lorsqu'un document source est modifié, la correspondance devient un publieur en envoyant des messages aux souscripteurs. Dans ce cas, les souscripteurs sont tous les acteurs identifiés comme **propriétaire** ou **contributeur** d'un document lié au document modifié. L'acteur est alors informé qu'un document utilisé pour créer son document a été modifié afin de vérifier si son document est impacté par la modification du document « publieur ».

4.6.2.3 Interface Automation

L'interface « Automation » permet aux utilisateurs de définir la procédure de génération des documents reliés à une TC. Cette interface contient une méthode « doGenerate ». Les objets implémentant cette interface vont avoir pour but de permettre aux utilisateurs de générer automatiquement les documents cibles de la correspondance. Cette action est possible si la correspondance est de type automatique, effectuée par un script ou un programme par exemple, et que l'acteur faisant cette transformation n'a pas à intervenir dans la génération. Il revient alors à l'acteur de définir tous les points permettant de générer les cibles (localisation des sources, localisation du programme, commande de lancement du programme…). Lorsque ces critères sont définis, et qu'une des sources de la transformation est modifiée et détectée, il est possible de prévenir l'acteur responsable des cibles qu'une source a été modifiée mais également, lui proposer de lancer la génération définie précédemment afin de propager automatiquement la modification. Cette technique permet de faire tendre la maintenance de cohérence vers l'automatisation et la systématisation de propagation de changements.

4.6.3 Propagation des changements

Le paquetage de **propagation** (Figure 4-8) impose les contraintes quant à la propagation des changements.

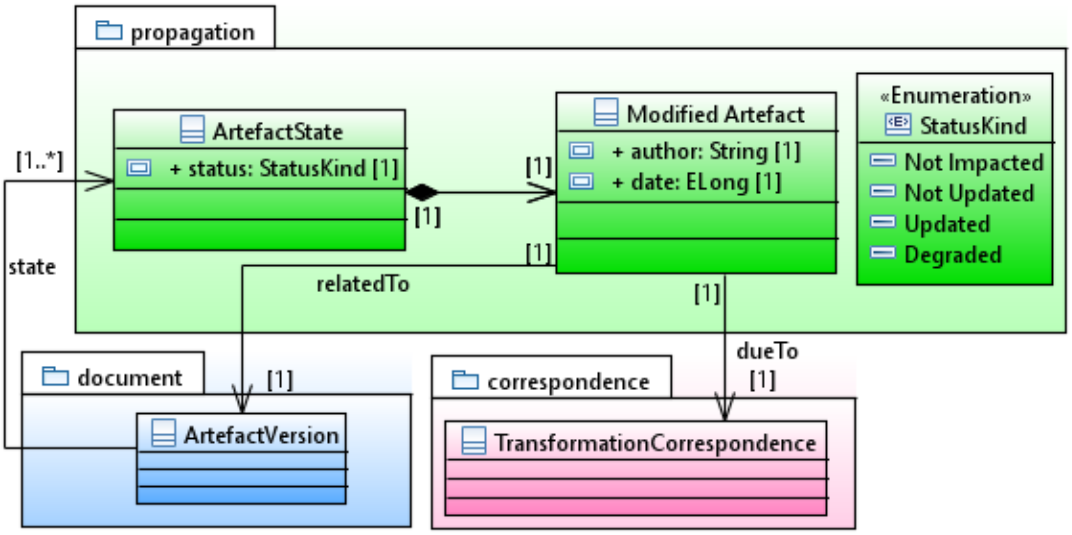


FIGURE 4-8 PAQUETAGE "PROPAGATION"

4.6.3.1 State

Une version de document représenté par un *artefact version* contient un ou plusieurs éléments *State*, représentant l'état de la version du document. L'état du document peut être de quatre types : *Not Impacted*, *Not Updated*, *Updated*, *Degraded*. L'état du document et son évolution sont déterminés grâce aux choix des acteurs sur le système. Les transitions entre états sont illustrées sur la Figure 4-9.

4.6.3.2 Status Kind

L'état *Not Impacted* d'un document enregistre le fait qu'un document lié à travers une *Transformation Correspondence* a été modifié mais que cette modification n'a eu aucun impact sur le document. Les deux documents sont donc toujours cohérents.

L'état *Not Updated* d'un document renseigne le fait qu'un document lié à travers une *Transformation Correspondence* a été modifié et qu'aucune décision n'a été prise quant au document. Cet état n'est qu'une transition.

L'état *Updated* d'un document signifie qu'un document lié par une *Transformation Correspondance* a été modifié. L'état du document est passé en *Not Updated*, l'acteur responsable du document a alors pris la décision de modifier son document afin de le maintenir cohérent avec la modification du document lié. Les documents sont donc devenus à nouveau cohérents.

L'état *Degraded* d'un document signifie qu'une *Transformation Correspondence* a volontairement été « cassée » par un acteur. Les documents impliqués dans la *Transformation Correspondence* ne sont plus liés. Cependant nous conservons une trace de la *Transformation Correspondence* au travers du *Modified Artefact* lié à l'état *Degraded*, identifiant ainsi l'acteur qui a dégradé le lien.

4.6.3.3 Modified Artefact

Un élément *State* est toujours relié à un élément *Modified Artefact*. Un élément *Modified Artefact* représente une modification. Il permet d'identifier quelle a été la transformation mettant l'*Artefact* dans cet état, ainsi que l'artefact ayant été modifié. L'état d'un artefact est toujours dépendant d'une modification qui est due à une correspondance et un artefact.

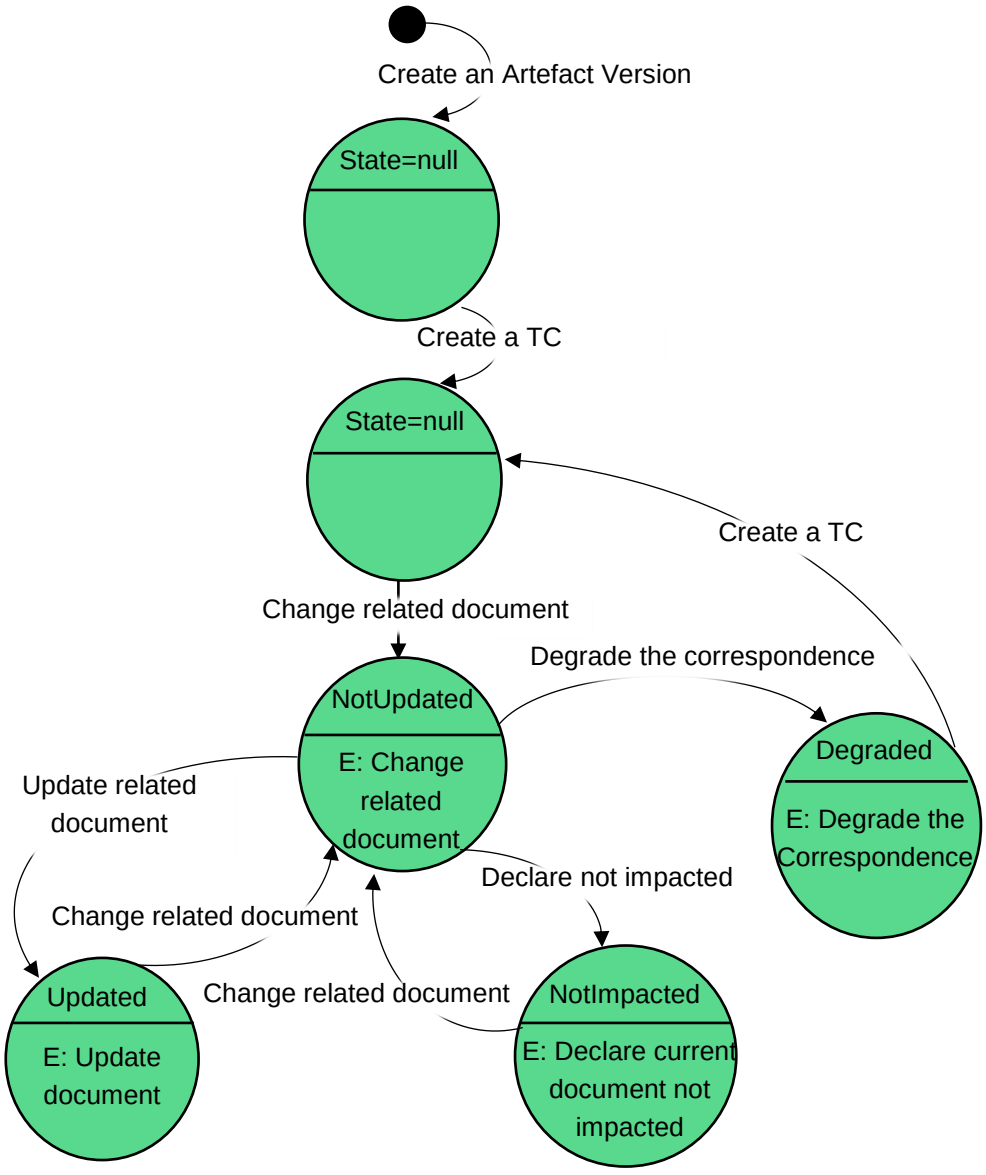


FIGURE 4-9 DIAGRAMME ÉTATS-TRANSITION DES ÉLÉMENTS *STATE*

4.7 Conclusion & Discussion

Ce chapitre présente l'approche proposée. Celle-ci se décompose en deux contributions majeures : une définition des liens entre les documents et l'exploitation de ces liens afin de maintenir la cohérence du corpus documentaire. Cette approche est basée sur le standard de description d'architecture ISO/IEC/IEEE 42010 et propose les concepts minimaux et nécessaires à la définition d'architecture et son maintien pendant le cycle de vie d'un système.

La définition des liens de cohérence se base sur le fait qu'une modification ou une création de document peut être vue comme une transformation. De ce fait, les caractéristiques des transformations peuvent être utilisées afin de définir complètement les liens entre les documents. Afin d'être la moins invasive possible, l'étape de création des liens est intégrée dans les pratiques industrielles classiques : lorsqu'un acteur publie un nouveau document, les méta-informations de ce document sont extraites afin de créer les liens.

L'approche permet de créer une traçabilité à n'importe quel niveau de granularité. Toutefois, nous défendons ici l'idée qu'une traçabilité au niveau « document » (qui lui-même a une granularité adaptable, car il peut regrouper plusieurs fichiers) est préférable. En effet, une granularité trop fine verrait le nombre de liens à maintenir tout au long du cycle de développement du système exploser.

L'exploitation des liens permet, d'une part, d'aider les acteurs du flot à propager les modifications intervenues sur un document de façon systématique. D'autre part, notre approche permet de faire émerger le processus réel extrait de l'observation des pratiques des acteurs. Outre l'utilisation par les cadres pour gérer efficacement le développement d'un système, son utilisation permettrait aux acteurs de connaitre le véritable impact d'une modification. De plus, dans une optique de Gestion des Processus métiers (BPM) [107], l'émergence du processus permettrait d'automatiser les phases de conception et de modélisation du processus.

Chapitre 5
Mise en œuvre de l'approche

Sommaire

Afin de pouvoir mettre en œuvre et tester l'approche, un prototype nommé APPE pour *Aided Propagation & Process Emergence* a été développé. Nous avons choisi de le développer en Java, en partie pour la portabilité de ce langage. En effet, notre application doit pouvoir être utilisée à la fois sur des postes Linux que sur des postes Windows, selon l'habitude des employés. Notre volonté en créant ce prototype est de mettre en œuvre toutes les fonctionnalités que l'approche avance. Le développement de ce prototype a également nécessité l'utilisation de tests de non-régression ainsi que de gestion de version.

Dans ce chapitre, nous commencerons par présenter les exigences fonctionnelles ayant dirigées le développement du prototype. Nous expliciterons ensuite le fonctionnement et la conception d'APPE.

5.1 Exigences fonctionnelles

Des exigences fonctionnelles ont guidé le développement du prototype. Nous donnons en Annexe 1 les exigences, résumées dans un tableau, que nous explicitons ici. Nous référençons entre parenthèse les exigences que l'on peut retrouver dans le tableau en annexe.

Le programme doit mettre en œuvre l'approche proposée, le point d'entrée est donc une publication effectuée par un utilisateur. Le programme doit être capable d'intercepter une

publication (APPE_010) afin d'en extraire les informations nécessaires à la création, la modification ou l'enrichissement d'une description d'architecture (APPE_020). Le programme doit avant tout être utilisable avec les dépôts de versions Subversion, ceux-ci étant pour le moment les plus utilisés par STMicroelectronics (APPE_011). Cependant, à terme, le programme devra pouvoir être utilisé avec n'importe quel gestionnaire de version (APPE_0112).

L'extraction des informations de la publication doit permettre d'identifier l'auteur d'un document (APPE_024), les contributeurs (APPE_025), ainsi que la version du document, sa localisation (APPE_26) et un horodatage. Cependant, toutes les informations nécessaires à la création d'une description d'architecture ne peuvent pas uniquement être extraites de l'analyse de la publication. En effet, la granularité des liens que nous avons choisie ne nous permet pas d'analyser le contenu des documents impliqués dans la publication.

Notre programme doit donc être capable de communiquer avec l'utilisateur au travers d'une interface graphique simple et utilisable par tous les acteurs du système (APPE_012) afin de récupérer des informations supplémentaires. Les informations supplémentaires doivent principalement porter sur la manière dont ont été créés les documents publiés. L'utilisateur doit pouvoir identifier les documents utilisés afin de créer les documents de la publication, contenant donc des informations dupliquées (APPE_023). L'utilisateur doit également pouvoir spécifier si les documents publiés ont été créés manuellement ou s'ils ont été générés grâce à un programme (APPE_021). Dans ce cas l'utilisateur doit pouvoir définir le programme utilisé et la ligne de commande nécessaire à son exécution (APPE_022).

Les informations extraites et saisies par un utilisateur doivent être sauvegardées selon le métamodèle défini dans le chapitre précédent (APPE_030) et être accessible aux autres utilisateurs (APPE_031). En effet, la description d'architecture doit pouvoir être partagée ou modifiée par tous les acteurs impliqués dans le développement d'un système sur puce. De plus, les informations doivent pouvoir être statistiquement analysées afin de générer automatiquement de nouveaux liens entre les documents (heuristiques [108]–[110]) (APPE_0322).

Lors de la détection d'une modification (APPE_040), la description d'architecture doit permettre au programme d'informer un utilisateur de l'impact de sa modification sur les autres documents (APPE_041). La description d'architecture doit également permettre à informer les utilisateurs potentiellement impactés de la modification d'un document contenant des informations communes (APPE_042) et proposer de lancer le programme permettant la génération des documents (si ce champ a été renseigné) (APPE_043). Enfin, la description d'architecture doit pouvoir être exploitée à chaque instant afin de visualiser le flot de conception des systèmes sur puce (APPE_032).

5.2 Aided Propagation and Process Emergence

Afin de répondre aux exigences, notre prototype a été développé en trois parties : crochet, client et serveur. La Figure 5-1 illustre l'architecture du prototype et la localisation de chaque partie.

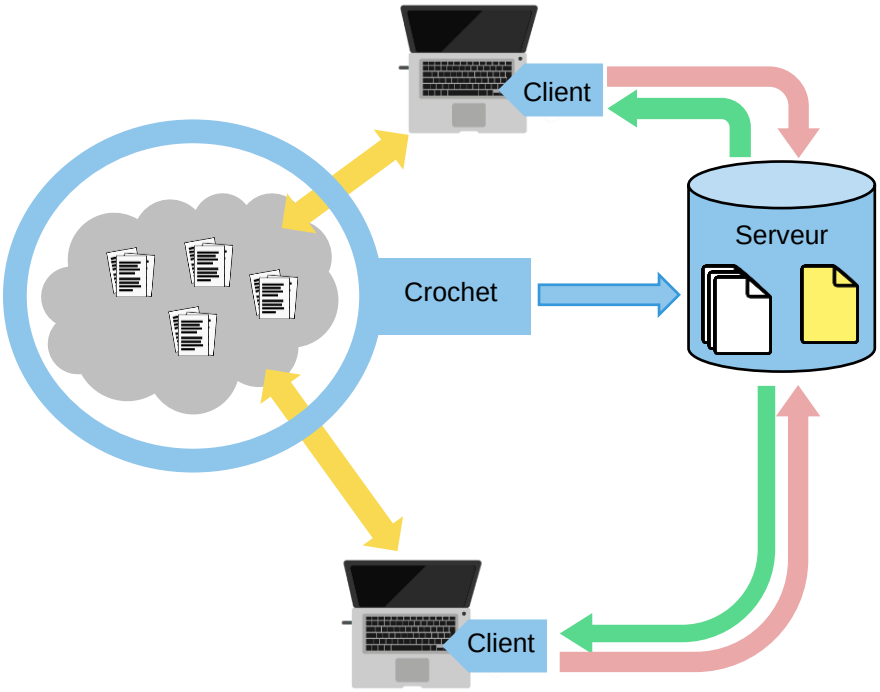


FIGURE 5-1 ARCHITECTURE DU PROTOTYPE APPE

La partie **crochet** est située sur le dépôt de versions (ici représenté par un nuage). Son but est d'intercepter les publications de documents effectuées par les acteurs, d'en extraire les méta-informations (documents impliqués, acteurs, identifiant de transaction) et de transmettre ces méta-informations au serveur. Le crochet répond au besoin de création des liens dans les pratiques de travail classiques présenté dans la Section 4.3.3.

La partie **serveur** est l'élément central de l'architecture de notre prototype. Il permet de stocker la description d'architecture et de l'enrichir grâce aux méta-informations récupérées par le crochet et celles données par l'acteur (au travers du client). Le serveur exploite donc à la fois le méta-modèle du standard ISO/IEC/IEEE 42010 et les extensions que nous proposons (Section 4.6). De plus, la partie serveur est chargée de propager les modifications, lorsqu'un changement est détecté, en avertissant par mail les acteurs potentiellement impactés.

Enfin, la partie **client** permet de faire le lien entre les acteurs et le serveur. Le client est situé sur le poste local de chaque acteur. Son but est de permettre aux acteurs de créer les liens (donner les informations sur la création des documents publiés) et d'informer les acteurs de l'impact potentiel de leur publication.

Nous allons maintenant voir en détail l'implémentation de ces trois parties.

5.2.1 Partie crochet

La partie crochet est décomposée en deux sous-parties : un script de pré-publication (*precommit script*) et un programme Java.

Le script de pré-publication fait appel à un mécanisme, nommé *hook*, disponible sur la plupart des logiciels de gestion de version (Subversion [111], Git [112], Mercurial [113]). Ces *hooks* sont des programmes lancés lors d'un évènement particulier (publication, verrouillage d'un document, déverrouillage d'un document …) intervenant sur un dépôt de version. Dans notre cas, nous avons implémenté un *pre-commit hook*. Lorsqu'une publication intervient sur le dépôt de version, le script est lancé. Celui-ci lit un fichier de configuration spécifiant où trouver Java, puis lance le programme en lui passant en paramètre le dépôt de version impliqué et l'identifiant de la transaction.

Le programme Java doit récupérer les informations de la publication et les transmettre au serveur. La Figure 5-2 représente le diagramme de classe simplifié de la sous-partie programme du crochet.

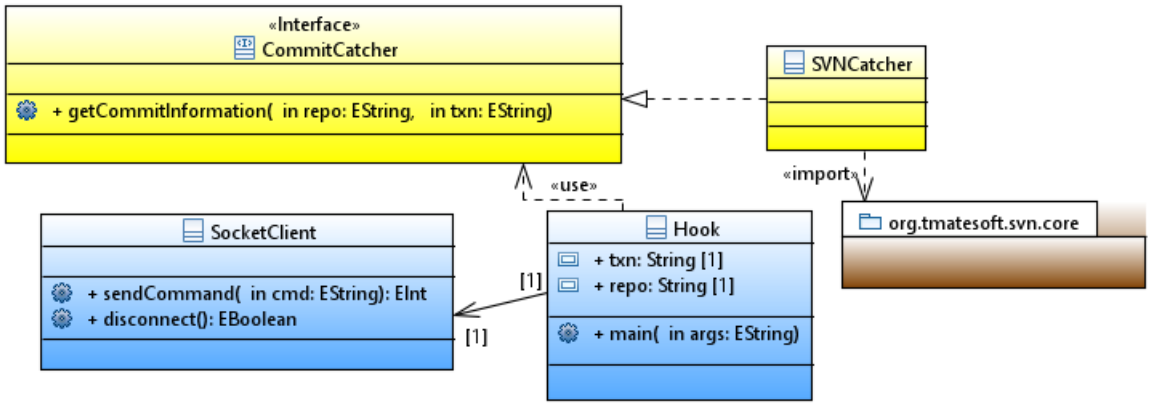


FIGURE 5-2 DIAGRAMME DE CLASSE SIMPLIFIÉ DE LA PARTIE CROCHET D'APPE

La classe *Hook* est le point d'entrée de la partie crochet. Son but est d'extraire les informations de la publication et de les envoyer au serveur. Cette classe utilise un objet implémentant l'interface *CommitCatcher*. Cette interface impose une méthode permettant de récupérer les informations de la publication (identifiant de l'acteur, liste des documents modifiés, message de publication, adresse du dépôt de version et identifiant de la transaction). Cette interface a pour but de rendre le crochet plus facilement adaptable aux différents logiciels de gestion de version. Dans notre cas, nous avons développé la classe *SVNCatcher* implémentant l'interface *CommitCatcher* pour récupérer les informations sur un dépôt Subversion. La classe *SVNCatcher* utilise la librairie Java **SVNKit** mise à disposition par TMate Software [114] permettant de manipuler les dépôt de version

Subversion. La classe *SocketClient* permet, quant à elle, de communiquer les informations extraites au serveur.

5.2.2 Partie Serveur

La partie **serveur** est composée d'un programme Java. Le but du serveur est de manipuler la description d'architecture (ajout, suppression ou modification d'information) et de gérer la propagation des modifications en notifiant les acteurs impactés par un changement. Le serveur est conçu pour être actif en permanence. La Figure 5-3 représente un diagramme de classe simplifié de la partie serveur.

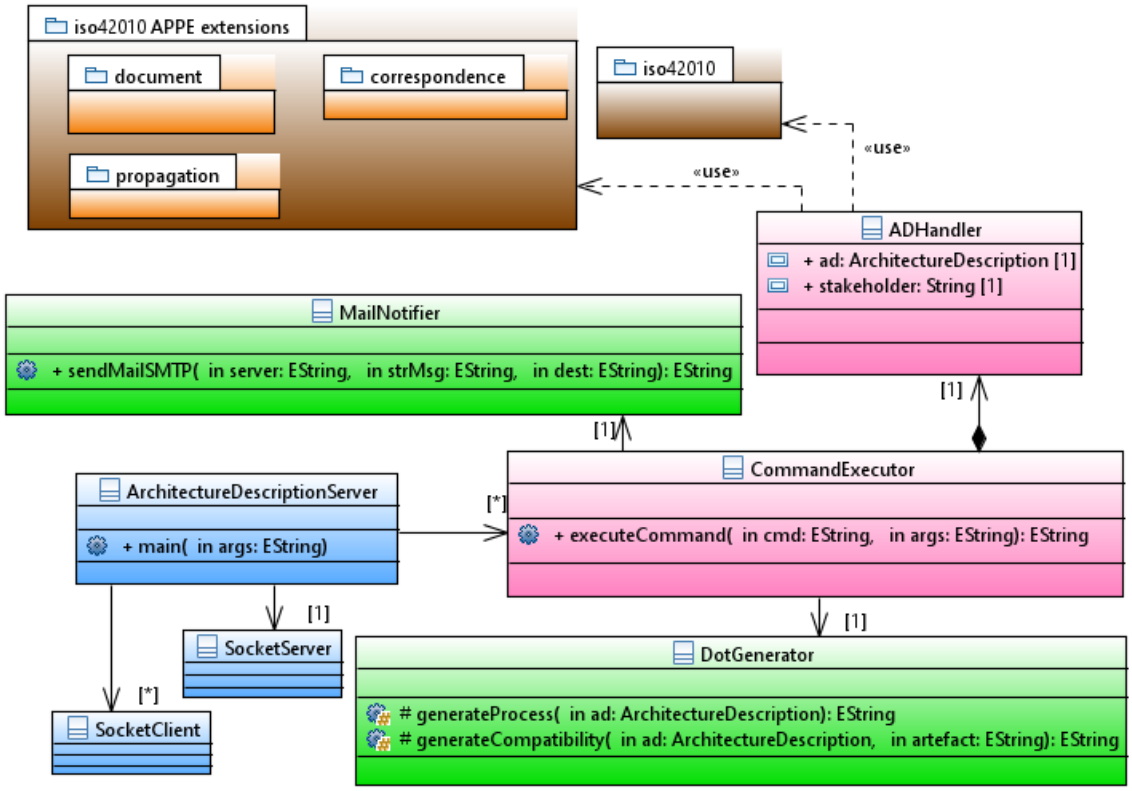


FIGURE 5-3 DIAGRAMME DE CLASSE SIMPLIFIE DE LA PARTIE SERVEUR D'APPE

Le point d'entrée du serveur est la classe *ArchitectureDescriptionServer*. Celle-ci possède deux *sockets* : un *SocketServeur* (afin de communiquer à la fois avec le crochet et le client) et un *SocketClient* (permettant de réveiller la partie client). La classe *ArchitectureDescriptionServer* peut utiliser plusieurs *CommandExecutor* (un par connexion active sur son socket). La classe *CommandExecutor* permet d'exécuter les commandes envoyées par le client. Nous donnons en Annexe 4 la liste des commandes traitées par le *CommandExecutor*. Le *CommandExecutor* gère l'envoi des mails de notifications aux acteurs grâce à la classe *MailNotifier* et la génération du processus au travers de la classe *DotGenerator* (génération d'un processus en *dot language* [115]). De plus, la classe *CommandExecutor* contient un objet *ADHandler* (pour *Architecture Description Handler*) permettant de manipuler et modifier les concepts de la description d'architecture. Pour ce

faire, la classe *ADHandler* utilise les métamodèles du standard ISO/IEC/IEEE 42010 (Annexe 3) et le métamodèle des extensions que nous proposons dans l'approche.

5.2.3 Partie Client

La partie **client** est composée d'un programme Java. Son but est de servir d'interface entre l'acteur et le serveur de description d'architecture. Le client est conçu pour tourner en permanence en tâche de fond sur le poste local des acteurs. La Figure 5-4 représente un diagramme de classe simplifié de la partie client.

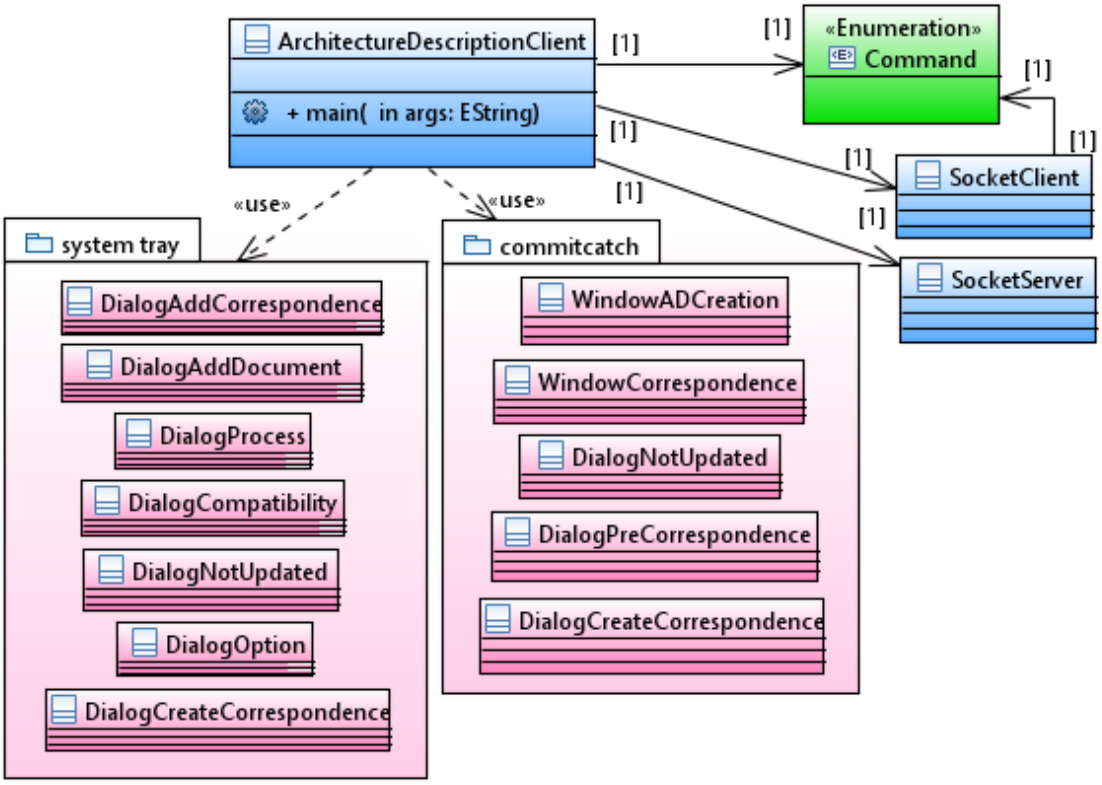


FIGURE 5-4 DIAGRAMME DE CLASSE SIMPLIFIÉ DE LA PARTIE CLIENT D'APPE

La classe *ArchitectureDescriptionClient* est le point d'entrée du client. L'*ArchitectureDescriptionClient* utilise deux sockets : un *SocketServer* permettant à la partie **serveur** de réveiller le client pour l'avertir d'une publication, et un *SocketClient* permettant au client d'envoyer des commandes (spécifiées par l'énumération *Command*) au serveur et de réceptionner les réponses. Le client utilise également de nombreuses interfaces graphiques permettant à un acteur (même non familier avec l'informatique) d'enrichir la description d'architecture.

5.2.4 Interaction entre les trois composants

La Figure 5-5 présente un diagramme de séquence permettant d'expliquer l'interaction précise des trois parties du prototype. Ce diagramme de séquence est séparé en deux phases : l'enregistrement et la publication.

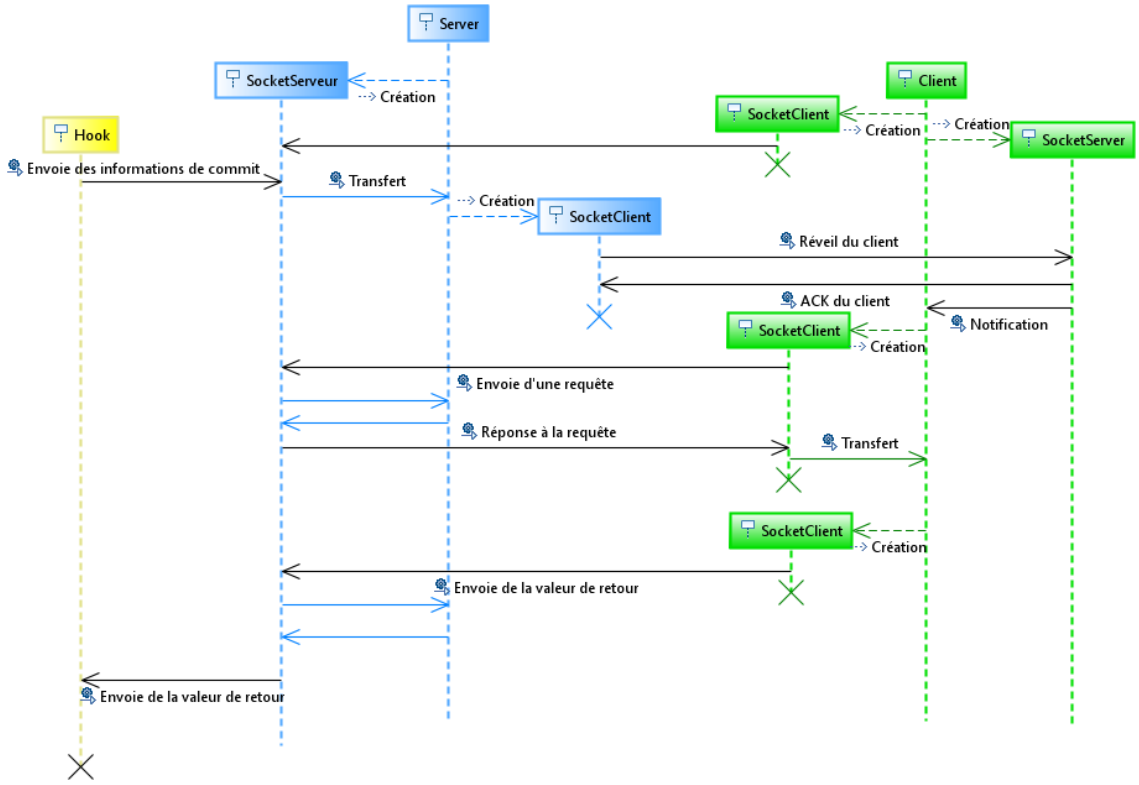


FIGURE 5-5 DIAGRAMME SÉQUENCE D'ENREGISTREMENT ET DE PUBLICATION

La phase d'enregistrement permet au client de se faire connaitre auprès du serveur et d'associer à chaque client un acteur spécifique. Lorsque le serveur est lancé, celui-ci se met en attente de connexion sur son socket. Lors du démarrage d'un client, un *SocketClient* est créé afin de transmettre au serveur l'adresse IP et l'identifiant de l'acteur. Le serveur stocke cette information et se remet en attente d'une communication sur son socket. Le client créé alors un socket et s'endort jusqu'à ce qu'une communication le réveille.

La phase de publication permet de gérer la publication de documents sur le dépôt de version. Lorsqu'une publication est effectuée par un acteur, le script de pré-publication suspend la publication et lance le crochet en lui fournissant en paramètres le dépôt de version impliqué et le numéro de transaction. Le crochet récupère l'identifiant de l'acteur et les documents contenues dans la publication, puis les transmet au serveur. Le serveur se réveille alors et crée un *SocketClient* pointant vers l'acteur spécifiquement désigné par l'identifiant afin de réveiller le client. Le client se réveille et confirme son éveil au serveur qui se déconnecte immédiatement, tuant ainsi le *SocketClient*. Le client affiche alors à l'acteur une interface graphique et se connecte au serveur afin de récupérer les informations à propos de la description d'architecture enregistrée. L'acteur peut alors interagir avec le serveur au travers du client pour modifier ou enrichir la description d'architecture. Lorsque l'acteur a terminé ses modifications, le client se charge de notifier le serveur de la fin de la communication. Le serveur renvoit alors une valeur de retour au crochet, permettant de spécifier si la modification s'est bien déroulée ou non. Selon cette valeur de retour, le crochet peut soit valider la publication, celle-ci se termine alors normalement et les modifications de

documents sont enregistrées dans le dépôt de versions, soit rejeter la publication, dans ce cas l'acteur est notifié grâce à son logiciel client Subversion usuel.

5.3 Conclusion

Dans ce chapitre, nous avons présenté le prototype que nous avons développé afin de mettre en œuvre l'approche. Ce prototype est séparé en trois parties : un crochet, permettant de récupérer les informations sur les publications, un serveur permettant de centraliser et de manipuler les descriptions d'architecture, et un client, permettant aux acteurs du flot de modifier ou d'enrichir les descriptions d'architecture (création des correspondances).

Le prototype se base sur l'utilisation du logiciel de gestion de version Subversion, car il est majoritairement utilisé dans STMicroelectronics. Cependant, l'utilisation de Git est de plus en plus importante. Un *CommitCatcher* permettant de récupérer les informations d'une publication Git pourra être développé ultérieurement. En effet, notre but lors du développement de ce prototype n'était pas de fournir un outil complet et utilisable en l'état, mais plutôt de vérifier la faisabilité de l'approche.

Bien que sous Git une publication se fait en deux temps (la première fois sur un dépôt local et la seconde sur le dépôt distant) et que les mécanismes avec Git sont différents de ceux de Subversion, nous pensons que l'approche et le prototype peuvent s'adapter très simplement. En effet, Git propose également le mécanisme de *Hook* permettant de lancer un script suite à un évènement particulier du dépôt. Il suffira donc d'implémenter un *CommitCatcher* adapté à Git (en utilisant par exemple la librairie JGit [116]). En ce qui concerne la publication en deux temps, il suffirait que le *hook* soit actif lors d'une publication sur le dépôt distant, lorsque la publication devient visible de tous les acteurs.

Chapitre 6
Application et évaluation de l'approche au travers de la création d'un sous-composant

Sommaire

Afin d'expliciter le fonctionnement et l'utilisation d'APPE et de notre approche, nous avons extrait, d'après une observation effectuée dans STMicroelectronics, un cas d'étude, simple, mais représentatif. Le processus décrit la création d'un *Intellectual Property block* (ou IP block) exploitable pour la simulation au niveau TLM. Ce cas d'étude sera également utilisé afin d'évaluer notre approche.

Dans ce chapitre, nous commencerons par présenter le cas d'étude en décrivant étape par étape les documents produits. Nous appliquerons ensuite notre approche à ce cas d'étude. Après avoir présenté en détail la méthodologie d'étude que nous avons conduite, nous développerons la conception et le déroulement de l'évaluation effectuée par les sujets. Enfin, nous dévoilerons les résultats obtenus lors de cette évaluation et nous évaluerons objectivement les résultats et les limitations de notre évaluation.

6.1 Description du cas d'étude de STMicroelectronics

Le cas d'étude présenté ici représente la création d'un IP block exploitable pour une simulation au niveau TLM. La Figure 6-1 représente le composant que nous souhaitons créer. Pour des raisons de confidentialité des IPs de STMicroelectronics, le composant présenté ici n'est pas un réel composant. Cependant, les sous-composants utilisés et les documents les décrivant sont extraits des vraies pratiques de travail.

Ce composant est composé de deux sous-composants : un UART (*Universal Asynchronous Receiver Transmitter*) et un DMA (*Direct Access Memory* ou Accès direct à la mémoire). Ces deux composants sont reliés et accessibles au travers d'un bus de communication. Le sous-composant UART contient six registres (partie accessible de la mémoire), tandis que le sous-composant DMA en contient 25.

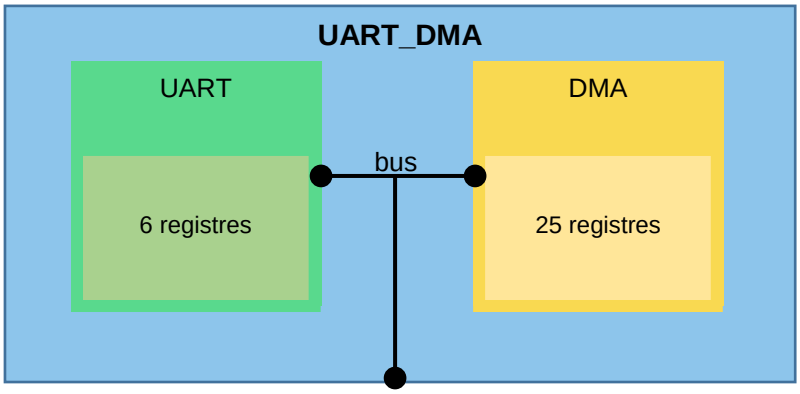


FIGURE 6-1 COMPOSANT UART_DMA

La Figure 6-2 représente le processus suivi pour créer ce block IP. Le processus est composé de 12 documents et de trois étapes de génération de documents.

Le processus commence par la spécification du bloc IP (uart_dma_system.docx). Cette spécification est rédigée en langage naturel par un membre de l'équipe d'architecture IP grâce à l'outil MS Word. La spécification décrit à la fois les aspects structurels et comportementaux de l'IP en détaillant les différentes entrées et sorties, les protocoles utilisés entre les instances de composant et l'adresse de chaque composant pour la structure. Le comportement est défini par la description des registres IP et la description des champs (détail de l'utilisation de chaque bit ou groupe de bits).

Cette spécification est ensuite analysée pour générer l'architecture structurale définie au format IP-XACT. La génération est réalisée par un outil propriétaire de STMicroelectronics: **spec2ipxact**. Dans notre cas, cet outil génère cinq documents décrivant la structure du composant :

- Deux descriptions des sous-composants (UART.xml et DMA.xml)
- Une description du composant entier (UART_DMA.xml)

- Une description des bus entre les sous-composants (UART_DMA_bus.xml)
- Une instance du composant (UART_DMA_design.xml)

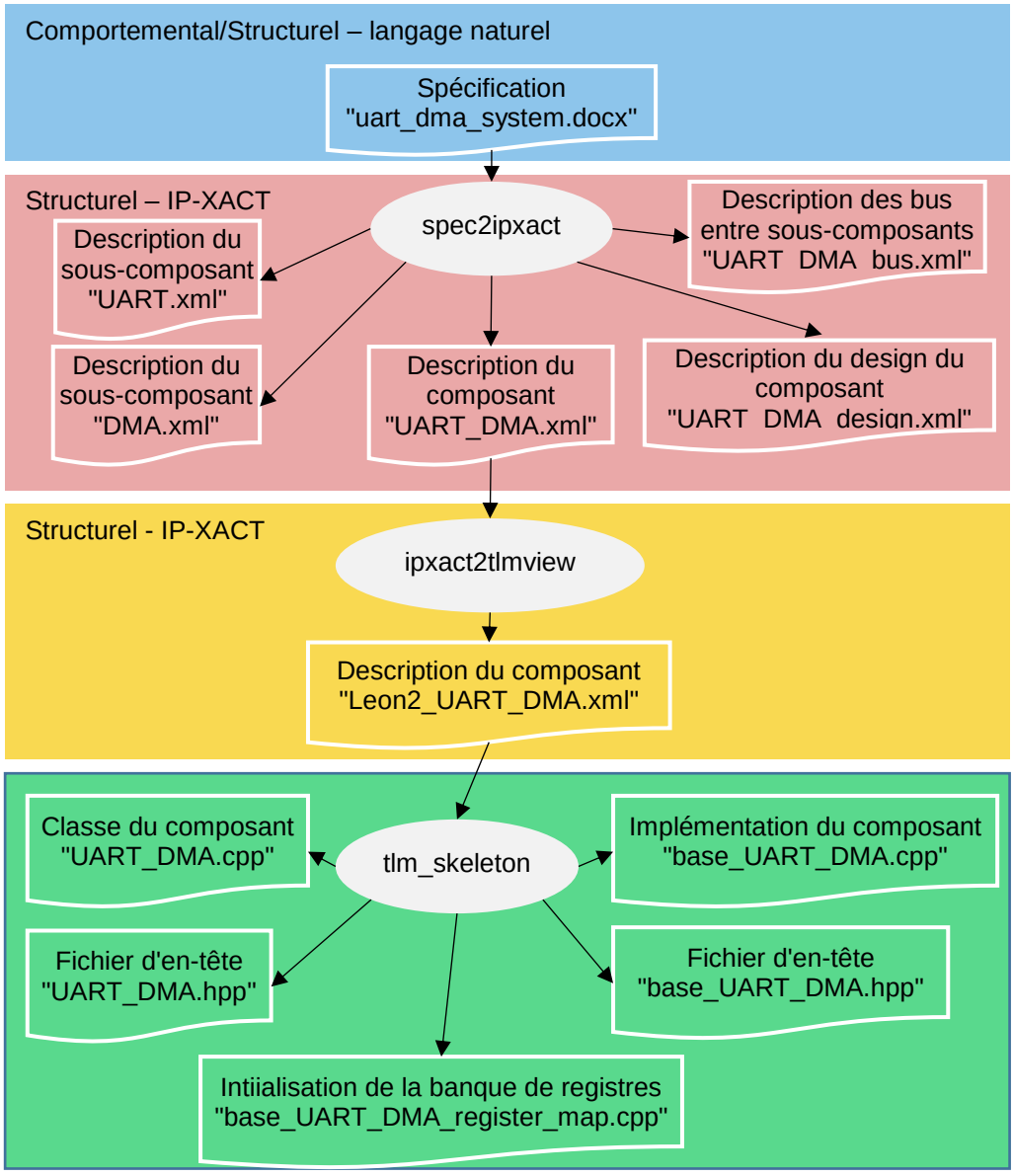


FIGURE 6-2 PROCESSUS DE CRÉATION D'UN IP BLOCK

Nous utilisons ensuite la description de l'architecture structurelle du composant (UART_DMA.xml) afin de générer la *TLM view* du composant. Cette génération se fait grâce à l'outil propriétaire **ipxact2tlmview** de STMicroelectronics. Le document produit (Leon2_UART_DMA.xml) est une abstraction de la description IP.

Ensuite, l'outil propriétaire **tlm_skeleton** de STMicroelectronics utilise le document TLM view pour générer l'architecture TLM du composant. L'architecture TLM est exprimée en SystemC (une bibliothèque C++ qui fournit des méthodes et des classes adaptées à TLM). L'architecture TLM générée est un squelette qui définit la structure de l'IP et que les

ingénieurs complèteront avec le comportement des composants. L'architecture TLM est composée de cinq documents :

- Une classe permettant l'implémentation du composant ("UART_DMA.cpp") et son fichier d'en-tête ("UART_DMA.hpp").
- Un document d'initialisation et d'implémentation du composant ("base_UART_DMA.cpp") et son fichier d'en-tête ("base_UART_DMA.hpp").
- Un document d'initialisation de la banque de registres ("base_UART_DMA_register_map.cpp")

Bien que ce processus semble très *top-down*, les acteurs sont amenés à modifier manuellement les fichiers générés, soit pour ajouter des informations manquantes, soit pour corriger des bogues.

6.2 Déroulement du processus avec APPE

Dans cette section, nous allons expliciter le flot précédemment présenté en utilisant APPE. Soient deux utilisateurs que nous appellerons "stakeholder1" et "stakeholder2" (respectivement S1 et S2). S1 est l'auteur de la spécification et de la partie SystemC, tandis que S2 est l'auteur de la partie IP-XACT. La partie automatisation d'APPE n'étant pas implémentée, nous supposerons que tous les documents sont créés manuellement.

6.2.1 Définition de la description d'architecture

La première phase que nous présentons ici tient lieu d'initialisation de la description d'architecture.

6.2.1.1 Première publication : Spécification

S1 crée la spécification du composant et la publie sur un premier dépôt de versions en utilisant son client SVN habituel. APPE détecte la publication et demande à S1 de choisir la description d'architecture (AD) qu'il souhaite enrichir. Vu que cette publication est la première, il n'existe aucune AD. S1 en crée une en spécifiant le nom de l'AD, ici "ArchitectureDescription_UART_DMA" (Figure 6-3).

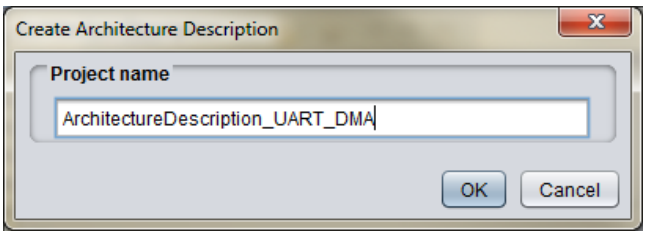


FIGURE 6-3 APPE S1: CRÉATION D'UNE DESCRIPTION D'ARCHITECTURE

Cette AD venant d'être créée, ne contient aucun document. S1 n'a donc pas besoin de créer de correspondances (Figure 6-4). En cliquant sur "OK", la publication se poursuivra normalement, et la spécification sera enregistrée dans le dépôt de version.

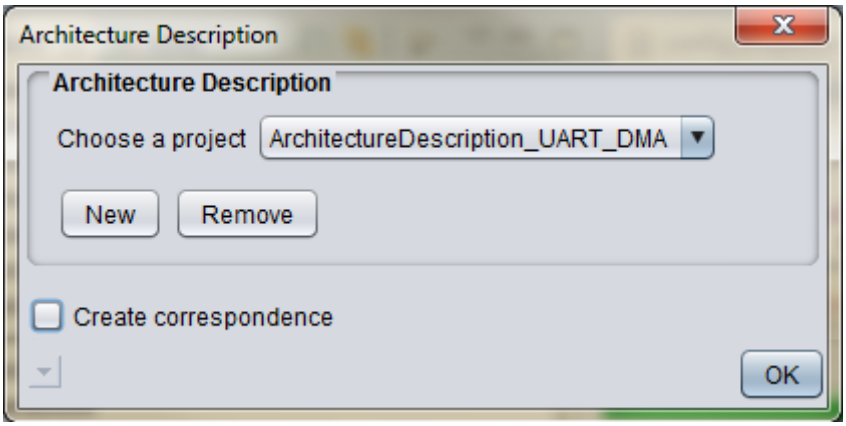


FIGURE 6-4 APPE S1: FENÊTRE PRÉ-CORRESPONDANCE

6.2.1.2 Deuxième publication : partie IP-XACT

S2 récupère la spécification de la même manière que dans le flot de conception usuel et crée les cinq documents représentant la description structurelle du composant en IP-XACT. En publiant ces nouveaux documents, APPE se lance. L'AD créée par S1 est visible par S2 (Figure 6-5). Cette fois-ci S2 choisi de créer des correspondances, car celui-ci a utilisé des informations de la spécification pour créer ses documents.

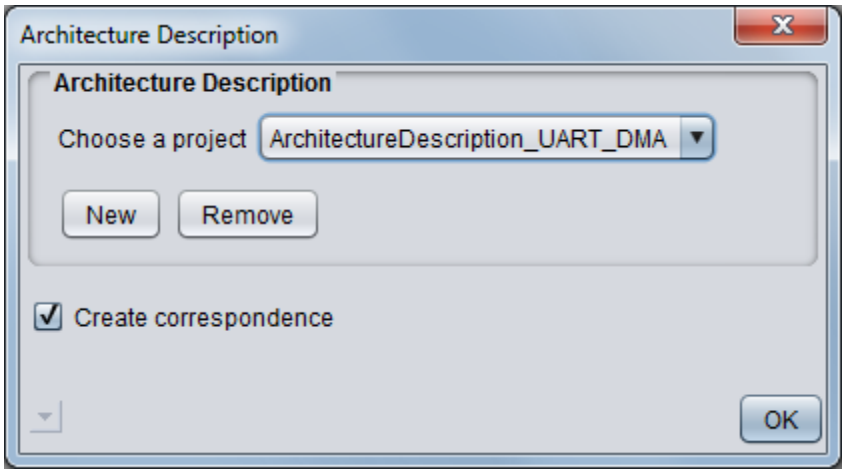


FIGURE 6-5 APPE S2: FENÊTRE PRÉ-CORRESPONDANCE

Une nouvelle interface graphique s'ouvre proposant un résumé de l'AD (Figure 6-6). Cette fenêtre est séparée en trois parties. La partie de gauche montre les documents dont les méta-informations ont été enregistrées dans l'AD. La partie du centre indique les correspondances dans lesquelles les documents sélectionnés à droite sont impliqués. La partie de droite affiche les documents impliqués dans la publication courante (ici nous retrouvons donc les cinq documents décrivant la structure du composant).

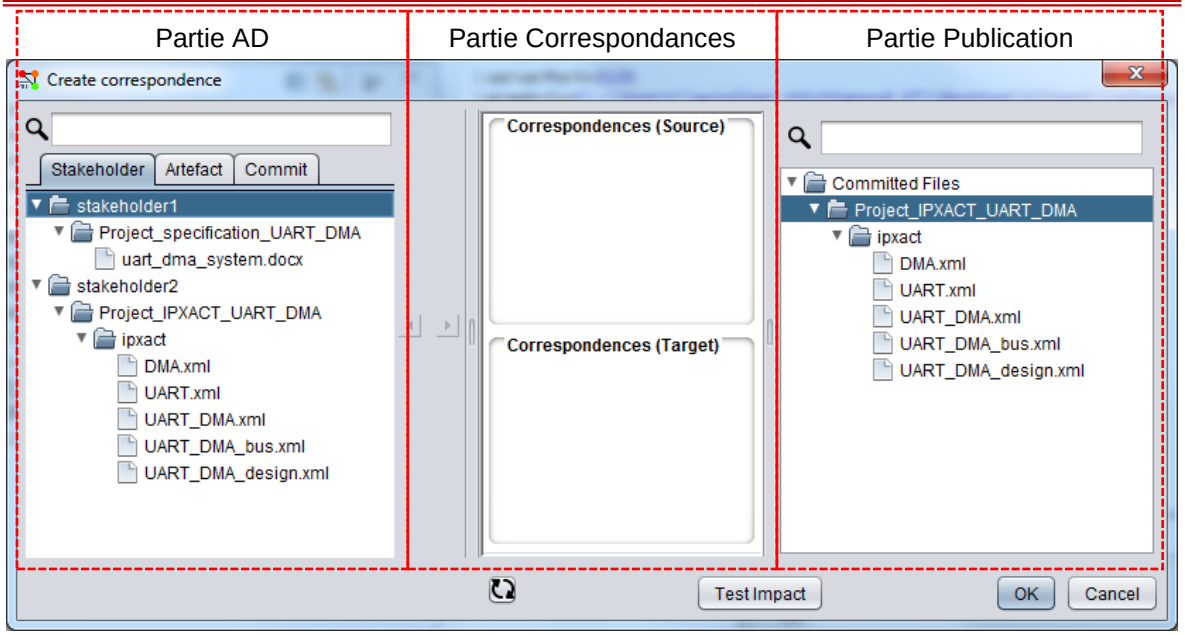


FIGURE 6-6 APPE S2 : FENÊTRE CORRESPONDANCE

S2 peut choisir de créer des correspondances simplement en sélectionnant les sources de la correspondance dans la partie de gauche (partie AD), les cibles de la correspondance dans la partie de droite (partie publication) et en cliquant sur la flèche à la gauche de la partie correspondance. Une fenêtre s'ouvre récapitulant les sources et cibles de la correspondance (Figure 6-7). Dans notre exemple, S2 crée une correspondance ayant pour source la spécification et comme cibles les deux descriptions de sous-composant (DMA et UART).

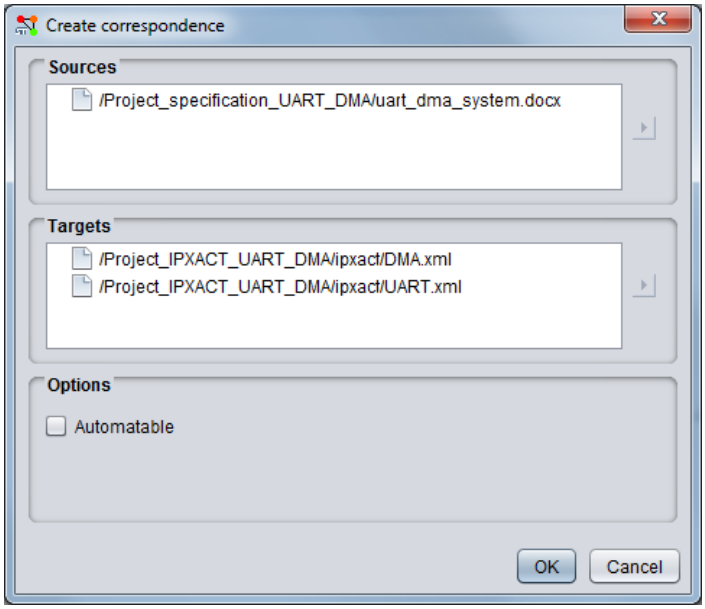


FIGURE 6-7 APPE S2: CRÉATION D'UNE CORRESPONDANCE

S2 crée deux autres correspondances avant de valider la publication : la première ayant pour sources les descriptions de sous-composant UART et DMA et la spécification et pour

cible la description du composant (UART_DMA), et la seconde ayant pour source la description du composant et pour cible la description des bus (UART_DMA_bus) et la description du design (UART_DMA_design).

6.2.1.3 Troisième publication : partie TLM/SystemC

S1 récupère à son tour les documents décrivant la structure du composant et publie à la fois le document représentant la vue TLM du composant (formulé en IP-XACT) et les documents SystemC. S1 crée quatre correspondances lors de la publication :

- Une correspondance ayant pour source la description du composant et en cible la vue TLM (Leon2_UART_DMA).
- Une correspondance ayant pour source la vue TLM et en cible les trois documents contenant du code System C (base_UART_DMA, base_UART_DMA_register_map et UART_DMA).
- Une correspondance ayant pour source le document de code SystemC (base_UART_DMA) et pour cible son fichier d'en-tête.
- Une correspondance ayant pour source le document de code SystemC (UART_DMA) et pour cible son fichier d'en-tête.

6.2.2 Émergence du processus

À la fin de ces trois publications, la description d'architecture du flot est définie. Les deux utilisateurs peuvent au travers d'APPE afficher le processus suivi (Figure 6-8, la figure complète est donnée en Annexe 6.1).

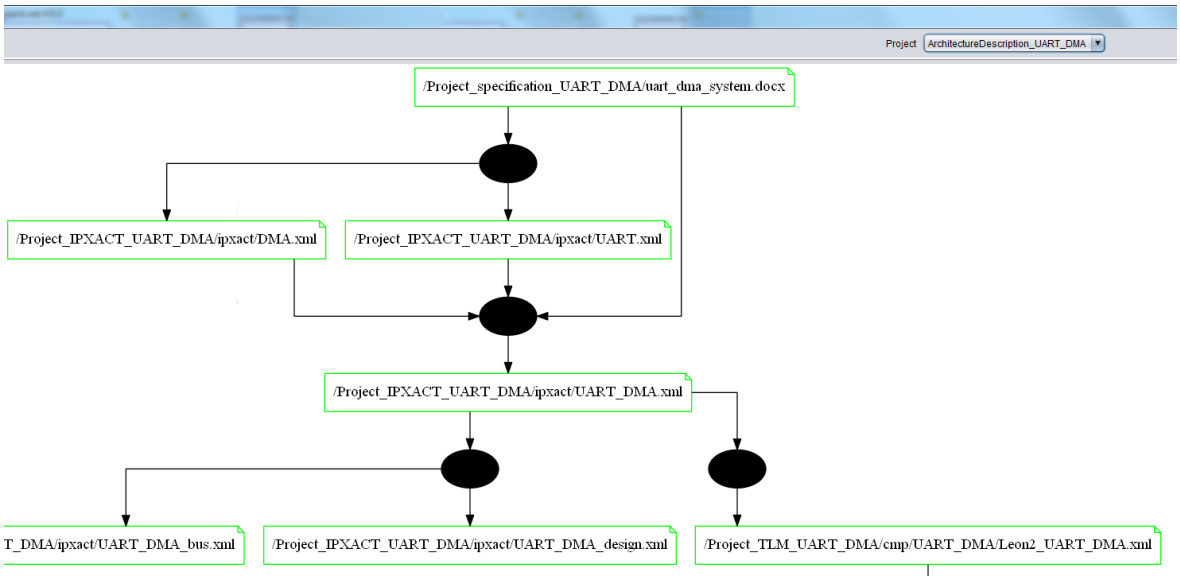


FIGURE 6-8 APPE : GÉNÉRATION DU PROCESSUS (EXTRAIT)

Ce processus est généré en *dot language* grâce aux informations contenues dans la description d'architecture. Chaque correspondance créee apparait sous la forme d'un ovale noir et chaque document impliqué dans une correspondance par un rectangle.

6.2.3 Propagation du changement

S2 décide maintenant de modifier la description du composant (UART_DMA.xml) afin de changer la taille d'un registre du sous-composant UART. Si l'on se réfère au processus (Figure 6-8), ce document est impliqué dans trois correspondances et six documents peuvent être potentiellement impactés. Tous les acteurs impliqués dans le développement de ces documents (tant les auteurs que les contributeurs) recevront un e-mail les prévenant d'un impact potentiel (Figure 6-9).

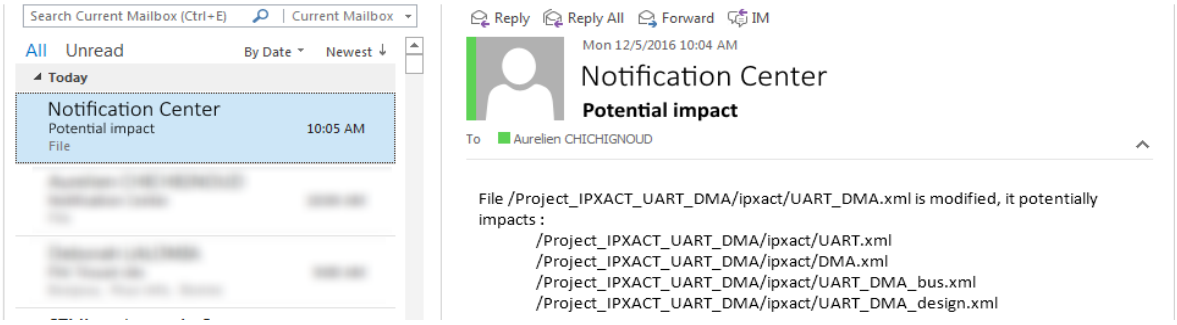


FIGURE 6-9 APPE S2 : NOTIFICATION MAIL D'UN POTENTIEL IMPACT

De plus, l'affichage du processus change afin de faire transparaitre ces impacts potentiels. Les rectangles représentant les documents passent du vert au rouge s'ils sont potentiellement impactés (Figure 6-10, la figure complète est donnée en Annexe 6.2).

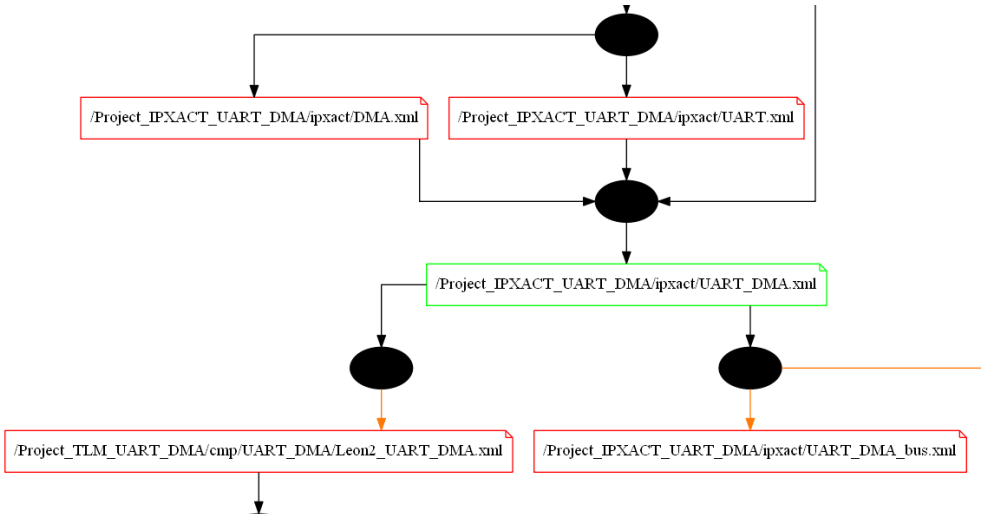


FIGURE 6-10 APPE : AFFICHAGE DU PROCESSUS LORS D'IMPACTS POTENTIELS (EXTRAIT)

Les utilisateurs peuvent également avoir accès à un récapitulatif des documents potentiellement impactés par le biais du client (Figure 6-11). Cette fenêtre est composée de deux parties. La partie de droite affiche une liste des documents nécessitant potentiellement une modification. Lors de la sélection d'un de ces documents, la partie de gauche affiche des informations sur le document ayant été modifié (date de la modification, acteur ayant effectué la modification, identifiant de la correspondance, message de publication).

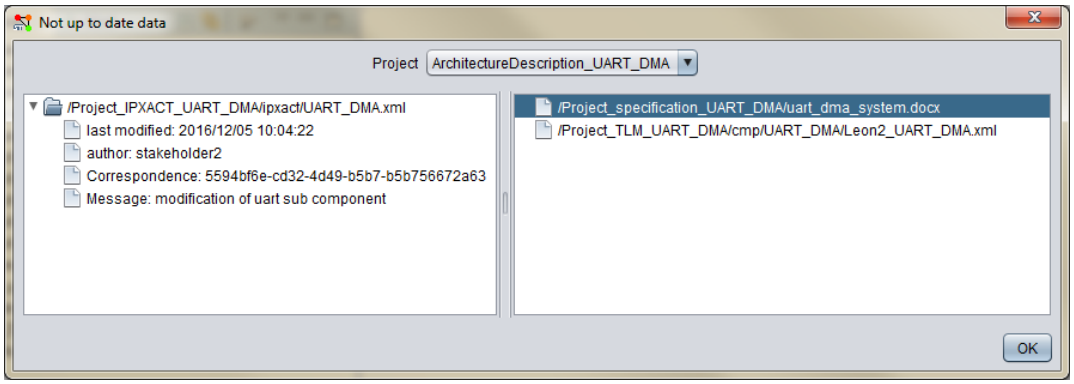


FIGURE 6-11 APPE S1: FENÊTRE RÉCAPITULATIVE DES IMPACTS POTENTIELS

À partir du message de publication ou en regardant la modification du document, l'utilisateur peut décider :

- Si son document n'est pas impacté.
- Si son document est impacté et nécessite une modification.
- Si son document est impacté mais il refuse de faire la modification.

6.2.3.1 Aucun impact

La modification de S2 portait sur le sous-composant UART dans la description du composant UART_DMA. Le document DMA est noté comme potentiellement impacté. Cependant sachant que la modification ne le touche pas directement, S2 décide d'annuler cet impact. Cette option est accessible dans la fenêtre récapitulative des impacts potentiels, en cliquant sur la source d'un impact potentiel (Figure 6-12). Les documents UART_DMA et DMA restent cohérents entre eux.

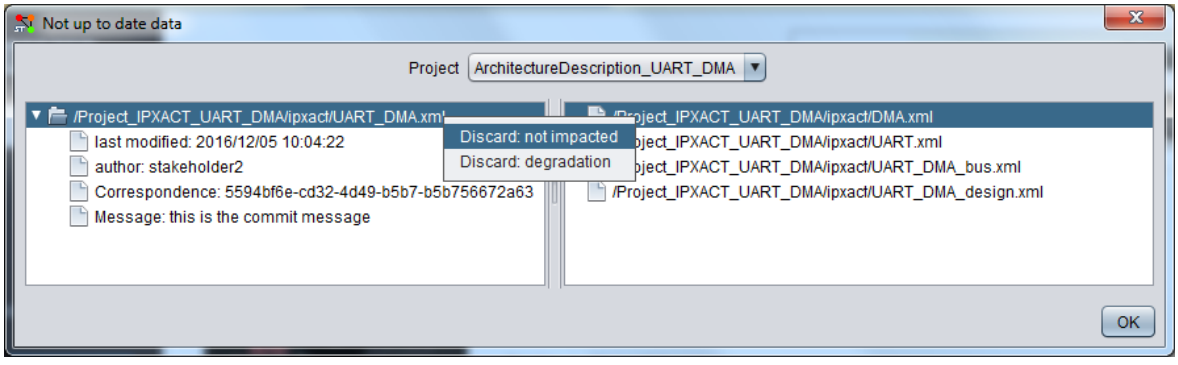


FIGURE 6-12 APPE S2 : AUCUN IMPACT

6.2.3.2 Mise à jour nécessaire

En revanche, le document décrivant le sous-système UART doit être mis à jour afin de rester cohérent avec la description du composant. S2 modifie donc son document et le publie. Cette fois-ci, lors du choix de la description d'architecture, APPE ouvre également une fenêtre permettant de préciser que ce commit répond à un besoin de mise à jour (Figure

6-13). En validant la publication, S2 va donc confirmer que les deux documents sont à nouveau cohérents entre eux.

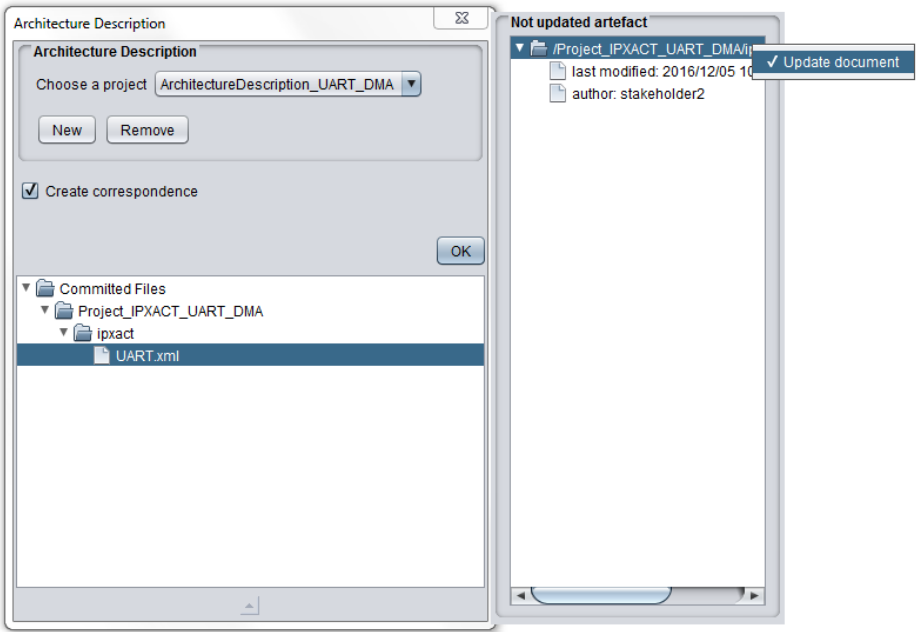


FIGURE 6-13 APPE S2 : MISE À JOUR DE DOCUMENT

6.2.3.3 Dégradation du flot

Le choix de dégrader le flot suite à une modification peut être fait dans la fenêtre récapitulative des impacts potentiels (Figure 6-12). Dans notre cas, S1 peut décider de dégrader le flot pour la vue TLM. Dans ce cas, la correspondance liant UART_DMA et Leon2_UART_DMA devient inopérante et disparait de l'affichage du processus (Figure 6-14, la figure complète est donnée en Annexe 6.3).

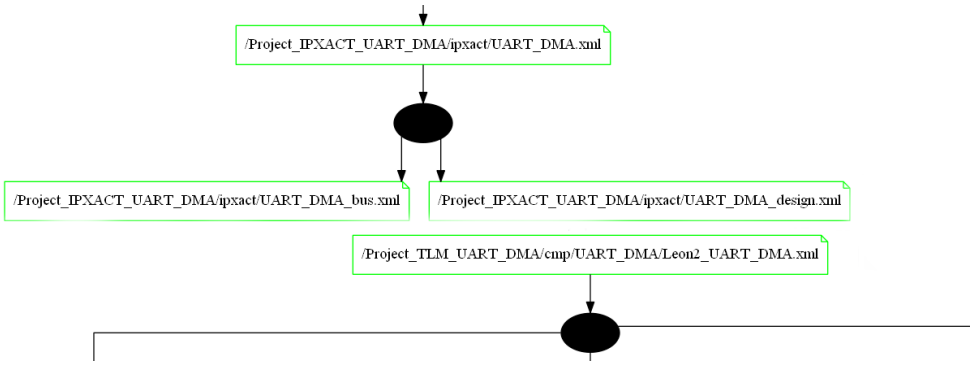


FIGURE 6-14 APPE S1 : DÉGRADATION DU FLOT (EXTRAIT)

6.2.4 Compatibilité des documents

Les trois options disponibles lors d'une modification d'un document que nous avons vu précédemment nous fournissent une information importante quant à l'interopérabilité des documents selon leurs versions. En effet, en sauvegardant tous les choix fait par les utilisateurs, il nous est possible de générer des arbres de compatibilité décrivant quels

documents sont cohérents selon la version utilisée (Figure 6-15 et Figure 6-16). Dans notre cas d'utilisation nous pouvons voir que l'utilisation de la description du composant version 2 obligera à utiliser également la seconde version de la description du sous-composant UART, et que la vue TLM est cependant incompatible car incohérente.

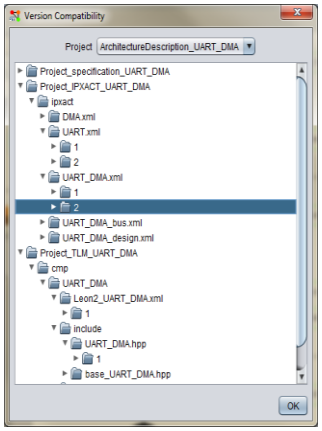


FIGURE 6-15 APPE : MODE DE COMPATIBILITÉ

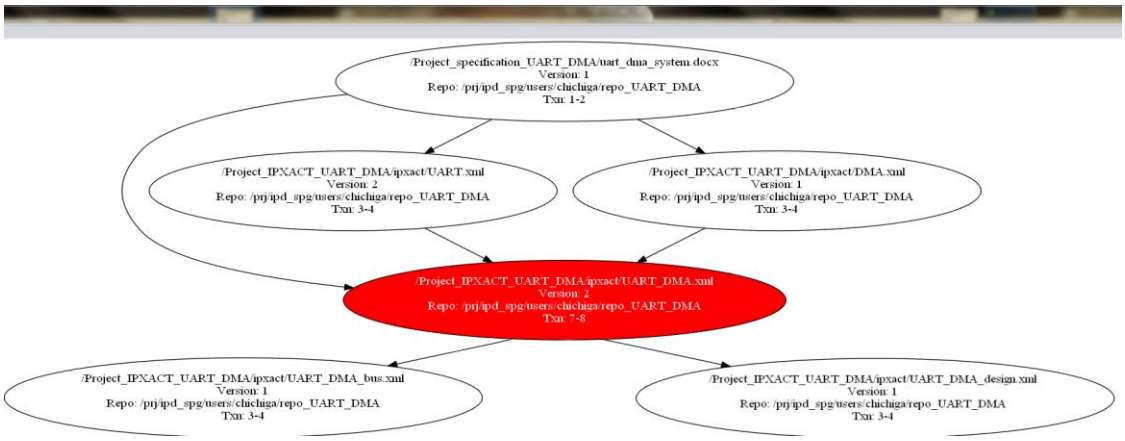


FIGURE 6-16 APPE : VISUALISATION DE LA COMPATIBILITÉ

6.3 Évaluation de l'approche

Outre l'avantage d'illustrer l'utilisation du prototype et de l'approche sur un exemple concret, nous utilisons le cas d'étude présenté dans la section précédente afin de vérifier la validité et l'efficacité de l'approche. Pour ceci, nous avons demandé à plusieurs sujets de se prêter à un test d'introduction d'incohérences. Nous avons mesuré deux variables, représentant le taux de détection et de correction d'incohérences et le temps moyen passé à corriger les incohérences, nous permettant de statuer quant à l'efficacité de l'approche.

6.3.1 Méthodologie d'étude

Notre évaluation s'appuie sur l'étude de données statistiques obtenues en effectuant plusieurs tests empiriques. Nous présentons dans cette partie, les différentes hypothèses que nous souhaitons tester.

6.3.1.1 But, questions et contexte

Nous formulons le but de notre étude en utilisant le modèle GQM (*Goal, Question, Metric)* [117]. Ce modèle a pour but de fournir un support permettant de clarifier les objectifs à atteindre (niveau conceptuel, *Goal*). Les différents attributs de ces objectifs sont identifiés par l'intermédiaire de questions (niveau opérationnel, *Question*). Enfin, des métriques sont associées à chaque question, afin d'y répondre de manière quantitative (niveau quantitatif, *Metrics*).

L'application du modèle est faite comme suit :

- Niveau conceptuel : Analyse des méthodes de propagation de changement dans le but de connaitre leurs impacts en matière de détection des incohérences du point de vue des développeurs dans le contexte d'un développement parallèle.

Nous pouvons en dégager deux questions :

- Q1: Est-ce que l'utilisation d'APPE améliore la facilité de détection et de correction des incohérences par les développeurs ?
- Q2: Est-ce que l'utilisation d'APPE a un effet sur le temps passé à corriger les incohérences ?

6.3.1.2 Variables et méthode de quantification

Les questions précédemment posées dans le cadre du modèle GQM nous mènent à définir plusieurs variables. Nous présentons ces différentes variables ici.

Pourcentage de correction :

Le pourcentage de correction a pour but de mesurer le nombre d'incohérences détecté et corrigé par les sujets. Cette variable représente le ratio entre le nombre d'incohérences corrigées par le sujet et le nombre total d'incohérences qui doivent être corrigées.

Effort :

La variable d'effort représente le temps moyen (en seconde) nécessaire à la correction d'une incohérence. Cette variable est calculée en divisant le temps passé à faire les corrections par le nombre de correction effectivement effectuée. Il faut noter que le temps est mesuré entre le moment ou le sujet récupère le fichier modifié et le moment où il publie ses modifications des autres documents.

6.3.1.3 Formulation des hypothèses

Hypothèse n°1 (H_1): La première question a pour but de savoir si les développeurs détectent et corrigent plus d'incohérences (dans un même espace de temps) en utilisant APPE qu'en utilisant seulement les méthodes de propagation usuelles. Généralement, les développeurs n'indiquent pas la présence ou l'existence d'incohérences. Les développeurs font la supposition que les décisions sur la conception sont correctes. De plus, les développeurs peuvent être forcés de produire un résultat même si des incohérences sont présentes. Notre intuition nous pousse à croire que les développeurs détectent et corrigent plus d'incohérences en utilisant APPE qu'en utilisant les méthodes usuelles de propagation, car les informations supplémentaires fournies par APPE leur permettent de cibler plus facilement les documents impactés par un changement. Cependant, l'effet inverse peut se produire ; les développeurs peuvent être parasités par des informations trop nombreuses fournies par APPE. Dans ce cas, les développeurs détecteront et corrigeront moins d'incohérences en utilisant APPE qu'en utilisant les méthodes de propagation usuelles. Nous résumons ces hypothèses par:

Hypothèse nulle 1, $H_{1\text{-}0}$: Les développeurs détectent et corrigent un pourcentage plus important ou le même pourcentage d'incohérence en utilisant seulement les méthodes de propagation usuelles qu'en utilisant APPE.

$H_{1\text{-}0}$: PourcentageDeCorrection (SansAppe) ≥ PourcentageDeCorrection (AvecAppe)

Hypothèse alternative 1, $H_{1\text{-}1}$: Les développeurs détectent et corrigent plus d'incohérence en utilisant APPE qu'en utilisant seulement les méthodes de propagation usuelles.

$H_{1\text{-}1}$: PourcentageDeCorrection (SansAppe) < PourcentageDeCorrection (AvecAppe)

Hypothèse n°2 (H_2): La deuxième question a pour but de découvrir si les développeurs passent plus de temps pour corriger le même nombre d'incohérences en utilisant les méthodes de propagation usuelles qu'en utilisant APPE. APPE se sert de correspondances créées en amont du processus pour savoir quels sont les développeurs qui doivent être notifiés d'un changement. Ces notifications contiennent une liste de documents potentiellement impactés par le changement qui doivent être vérifiés. Nous supposons que les informations données par APPE vont permettre aux sujets de corriger plus rapidement les incohérences, grâce aux informations supplémentaires cadrant le changement et son impact. Il n'est pas exclu que les informations supplémentaires peuvent forcer les sujets à vérifier plus de documents et donc à perdre du temps pour identifier quels documents sont réellement impactés. Ceci nous mène à reformuler ces hypothèses comme suit:

Hypothèse nulle 2, $H_{2\text{-}0}$: Les sujets passent autant ou moins de temps à corriger les incohérences en utilisant seulement les méthodes de propagation usuelles qu'en utilisant APPE.

$H_{2\text{-}0}$: Effort (SansAppe) ≤ Effort (AvecAppe)

Hypothèse alternative 2, $H_{2\text{-}1}$: Les développeurs passent plus de temps à corriger les incohérences en utilisant seulement les méthodes de propagation usuelles qu'en utilisant APPE.

$H_{2\text{-}1}$: Effort (SansAppe) > Effort (AvecAppe)

6.3.1.4 Conception de l'étude

Ces différentes hypothèses doivent être testées au cours de plusieurs tests. Nous développons ici l'étude et les tests qui ont été conduits afin de pouvoir juger les différentes hypothèses.

Choix des sujets :

18 sujets volontaires ont été sélectionnés pour évaluer APPE. Les sujets de l'expérimentation sont issus de catégories variées : étudiants, professionnels familiers ou non avec le développement de système sur puce. Ces sujets ont été sélectionnés pour leurs connaissances de l'informatique et leurs capacités à effectuer toutes les tâches nécessaires au déroulement de cette étude. Les sujets ont les connaissances nécessaires pour utiliser les différents éditeurs de documents et sont familiers du contrôle de version avec Subversion. De plus, les sujets n'ont pas été formés à l'utilisation d'APPE et ne sont pas familiers du corpus documentaire.

Nous avons volontairement choisi de ne pas solliciter uniquement des développeurs de STMicroelectronics. Lors de la conception de l'évaluation, une restructuration des différentes divisions était en cours et n'a pas permis d'effectuer une évaluation à plus grande échelle par les développeurs de STMicroelectroncis.

Comparaison des deux tests :

Tous les sujets sont soumis à deux tests : avec et sans APPE. Chaque test implique le traitement par le sujet d'incohérences ajoutées dans un ensemble de documents. Pour le premier test, les sujets ont accès aux informations et aux notifications fournies par APPE pendant la détection des changements. Pour le second test, les sujets ont seulement accès aux commentaires de publication de changements décrivant quelle partie du document est modifiée et quel est le sens de la modification. En ce qui concerne le test avec APPE, la description de l'architecture (donc les correspondances entre les documents) est créée avant le test. Les incohérences ajoutées en modifiant les documents sont différentes pour chaque test afin de minimiser l'apprentissage des relations entre les documents et éviter l'influence du premier test sur le second.

Tâches :

Pour chaque test, les sujets sont responsables d'un ensemble de douze documents partageant des informations entre eux. Le corpus documentaire est composé d'une spécification au format MS Word, six descriptions de composants au format IP-XACT, trois fichiers de code sources C++ et deux headers (Voir Chapitre 6 pour plus de détail).

Afin de simuler un travail distribué, toutes les deux minutes un document est modifié par l'opérateur de l'expérimentation. Par conséquent, des incohérences sont ajoutées dans le corpus documentaires. Les sujets sont alors sollicités pour modifier les documents impactés afin de garder l'ensemble cohérent. Pour chaque test, neuf documents sont modifiés. Afin d'être tout à fait équitable, une attention particulière a été portée aux modifications apportées aux documents afin que les deux tests impliquent un nombre égal d'incohérences à corriger, fixé ici à 63 incohérences induites. Les sujets devaient publier leurs modifications aussitôt qu'ils sentaient la propagation terminée, dans un délai maximal de deux minutes.

6.3.2 Résultats d'expérience

Tous les résultats obtenus lors de nos tests sont données en Annexe 7. Nous présentons en Table 6-1 un résumé descriptif des résultats de l'étude. En Table 6-2, nous donnons les résultats des tests statistiques effectués : test de Shapiro-Wilk [118] et test de Student apparié [119]. Le test de Shapiro-Wilk teste l'hypothèse nulle selon laquelle un échantillon est issu d'une population normalement distribuée. Tandis que le test de Student apparié a pour objectif de tester l'hypothèse d'égalité de l'espérance de deux variables aléatoires suivant une loi normale.

TABLE 6-1 STATISTIQUES DESCRIPTIVES DES MESURES EFFECTUÉES

Variable	Test	Moy.	Écart-type	Min.	Méd.	Max.	Différence (en %)
Détection/ Correction	Sans APPE	0,76	0,11	0,52	0,75	0,92	5.5
	Avec APPE	0,80	0,13	0,56	0,80	0,98	
Effort	Sans APPE	18,7	0,97	16,5	18,8	20,1	3.9
	Avec APPE	18	1,02	16,5	18	19.9	

TABLE 6-2 TESTS STATISTIQUES DES MESURES EFFECTUÉES

		Shapiro-Wilk	Student	
Variable	Test	W_{obs}	t	p-value
Détection/Correction	Sans APPE	0,957	3,77	0,00153
	Avec APPE	0,955		
Effort	Sans APPE	0,958	3,72	0,00179
	Avec APPE	0,95		

Avec α=0,05, W_{crit}=0,897

6.3.2.1 Détection/Correction d'incohérences

Analyse descriptive. La première question avait pour but l'étude de la pertinence d'APPE sur la détection et la correction d'incohérences par les sujets. Les sujets détectent et corrigent plus d'incohérences en utilisant APPE qu'en utilisant les méthodes de propagation usuelles. Ce taux supérieur de détection et de correction peut être vu dans la Table 6-1 en comparant les moyennes et les médianes. Les sujets détectent et corrigent en moyenne 5,5% d'incohérences en plus avec APPE qu'avec les méthodes de propagation usuelles.

Analyse Statistique. Le test de Shapiro-Wilk indique que les données sont normalement distribuées ($W_{obs} > W_{crit}$), un test de Student peut donc être appliqué afin de tester l'hypothèse 1. La *t-value* est de 3,77 avec une *p-value* de 0,00153 (Table 6-2). Cette *p-value* (<0.05) nous indique que l'hypothèse nulle 1 (PourcentageDeCorrection SansAppe ≥ PourcentageDeCorrection Avec APPE) peut être rejetée ; la différence moyenne de détection et de correction d'incohérences avec et sans APPE n'est pas nulle. Par conséquent, nous avons de forte preuve que les sujets détectent et corrigent plus d'incohérences en utilisant APPE qu'en utilisant les méthodes de propagation usuelles.

6.3.2.2 Effort

Analyse descriptive. La deuxième question avait pour but l'étude du temps passé à corriger les incohérences par les sujets avec et sans APPE. Les sujets utilisent moins de temps pour corriger les incohérences en utilisant APPE qu'en utilisant les méthodes de propagation usuelles. Ce taux inférieur d'effort à fournir pour corriger les incohérences peut-être vu dans Table 6-1 en comparant les moyennes et les médianes avec et sans APPE. Les sujets corrigent les incohérences, en moyenne, 3,9% plus rapidement en utilisant APPE qu'en utilisant les méthodes de propagation usuelles.

Analyse statistique. Le test de Shapiro-Wilk indique que nos données sont normalement distribuées ($W_{obs} > W_{crit}$), un test de Student peut donc être appliqué pour tester l'hypothèse 2. La *t-value* est de 3,72 avec une *p-value* de 0,00179 (Table 6-2). Cette p-value (<0.05) nous indique que l'hypothèse nulle 2 (Effort sans APPE ≤ Effort avec APPE) peut être rejetée ; la différence moyenne de temps passé à corriger les incohérences avec et sans APPE n'est pas nulle. Par conséquent, nous avons de fortes preuves que les sujets passent moins de temps à corriger les incohérences en utilisant APPE qu'en utilisant les méthodes de propagation usuelles.

6.4 Conclusion de l'étude

Cette étude donne des résultats probants. En effet, nous avons pu constater que l'utilisation d'une description d'architecture et de correspondances permet d'améliorer la cohérence du corpus documentaire, impliqué dans le développement d'une IP, plus rapidement. Les sujets ont ainsi pu détecter et corriger plus d'incohérences en utilisant

l'approche qu'en utilisant uniquement les méthodes de propagation usuelles. En moyenne, les sujets ont ainsi corrigés 5,5% d'incohérences en plus en utilisant APPE. De plus, l'étude sur le temps passé pour corriger les incohérences nous informe que les sujets corrigent les incohérences, en moyenne, 3,9% plus rapidement en utilisant APPE qu'en utilisant les méthodes de propagation usuelles.

L'outillage que nous avons développé et qui met en œuvre l'approche proposée a été utilisé dans cette étude. Bien qu'encore à l'état de prototype, il nous a permis de démontrer l'apport que pouvait donner l'approche à la détection et la correction d'incohérences dans un environnement industriel. Cependant, notre étude présente tout de même quelques limitations discutées dans cette section.

Tout d'abord, les conditions de réalisation de l'expérimentation reposant sur des humains, il est possible que certains facteurs non mesurables aient biaisés les résultats (stress, fatigue…). Toutefois, les p-valeurs étant très faibles, nous avons décidé de négliger ce biais.

Ce cas d'étude très cadré ne nous permet pas de vérifier complètement la flexibilité de l'approche et sa mise en œuvre à grande échelle. En effet, lors de cette étude les différents sujets étaient seuls à faire des modifications. Afin de simuler un travail distribué, l'opérateur de l'expérimentation introduisait des incohérences dans le corpus documentaire. Cependant, nous ne pouvons pas juger de la qualité de cette simulation ni de la cohérence du système si un grand nombre de personnes interviennent dans la liste de correspondances. Une seconde évaluation serait nécessaire afin de statuer sur la mise à l'échelle de l'approche dans un cadre industriel. Il serait alors nécessaire d'avoir un nombre de documents plus importants, chaque sujet aurait alors une sous-partie de cet ensemble de document à maintenir cohérente, toutes les sous-parties seraient liées entre elles ; la modification par un sujet d'un sous-ensemble impliquerait qu'un autre sujet doive à son tour modifier son sous-ensemble de documents. Les changements et la propagation de ces changements se succèderaient alors en cascade entre les sujets. Nous pourrions ainsi juger si la propagation des changements et la stabilisation de la cohérence du système se fait correctement.

De plus, pour cette étude, la description d'architecture a été définie avant l'exécution concrète de l'évaluation afin de pallier à la méconnaissance du flot par les sujets. Cependant, la description de l'architecture par les sujets et par l'observation de leurs méthodes de travail fait partie intégrante de l'approche. Dans le cas présent, nous ne pouvons pas juger si l'utilisation de toutes les fonctionnalités d'APPE (comme la création ou la modification des correspondances) se révèle ergonomique et aisée. L'évaluation de cette partie de l'approche, devrait être effectuée en l'appliquant dans un projet en production dans STMicroelectronics, et devra être suivi par un entretien avec les sujets afin de pouvoir améliorer l'outil et l'approche en fonction de leurs avis.

Lors de cette évaluation, la description d'architecture a été créée avant d'effectuer les tests, afin de pallier à la méconnaissance de l'outil par les sujets. Cependant, décrire une

architecture demande un effort, tant intellectuel que temporel, qui n'a pas été étudié ici. Toutefois, l'approche cherchant à distribuer la création de la description d'architecture entre tous les acteurs, nous avons émis l'hypothèse que cet effort est négligeable.

Enfin, la correction d'incohérences en amont du flot de conception peut apporter un bénéfice indirect en faisant gagner un temps substantiel à la phase d'intégration des différents éléments se retrouvant dans un système sur puce. Par exemple, lors de l'intégration des sous-composants dans un composant, l'espace mémoire occupé par chacun des sous-composant est identifié dans une section de la spécification appelée *address map.* L'implémentation qui en est fait suit cette *address map*. Cependant, il peut arriver que celle-ci soit modifiée par l'architecte, sans que les développeurs en aval ne soient prévenus. Une telle modification entraine les développeurs à tenter d'accéder à des espaces mémoires incorrects. En phase de débogage, ce type d'incohérence peut provoquer plusieurs semaines de retard pour trouver et comprendre l'origine de l'erreur. L'utilisation de l'approche permettrait de corriger ou de prévenir cette incohérence en amont du flot, le bénéfice indirect étant alors le gain de temps non dépensé par ce débogage.

Chapitre 7
Conclusion & perspectives

7.1 Bilan

Cette thèse cherche à répondre à un enjeu industriel concret, s'approcher d'un système sans défaut, par l'élimination des incohérences afin de réduire le temps de développement. Nous nous sommes basé sur le domaine des semi-conducteurs, cependant, l'approche proposée est adaptable à tous domaines incluant le développement de systèmes complexes.

La première partie de ce manuscrit résume le raisonnement qui nous a menés à réaliser notre contribution. Notre analyse du flot de conception des systèmes sur puce (Chapitre 2) et des moyens utilisés aujourd'hui afin de s'affranchir en partie des spécificités de ce flot (Chapitre 3) a mis en évidence le besoin de maintenir systématiquement la cohérence des documents impliqués dans le flot. En effet, nous avons vu que le développement d'un système sur puce nécessite la mobilisation d'un grand nombre d'intervenants. Chacun de ces intervenants va venir enrichir la description du système en ajoutant des informations correspondant à son cœur de métier et exprimant un point de vue particulier sur le système. Ces informations sont encodées dans différents formats permettant à l'acteur d'effectuer son activité sur le système (analyse d'architecture, analyse de performance, validation, développement matériel ou logiciel …).

Cependant, ces documents décrits dans différents formats sont implicitement liés entre eux. En effet, la modification d'un document est susceptible de nécessiter la modification d'autres documents afin de garder l'ensemble du corpus cohérent. Ces liens ne sont généralement pas formulés.

Les incohérences des données de conception sont une source importante de bogues lors du développement du système. Elles impactent directement la capacité des équipes de développement à finaliser le produit avant sa mise en production en volume. Sachant que le prix d'un masque permettant la gravure de la puce sur le silicium peut atteindre plusieurs millions d'euros, il est capital qu'aucune ambigüité ou incohérence ne puisse devenir une source d'erreur une fois la puce produite. De plus, il est très important que la documentation livrée aux clients finaux soit cohérente avec le produit. Dans le contexte économique et concurrentiel propre au domaine de l'électronique et des semi-conducteurs, maintenir la cohérence du corpus documentaire impliqué dans le flot de conception des systèmes sur puce

est donc primordial pour garantir des produits de qualité, conserver un bon *time to market* avec un haut niveau de parts de marché et un bon prix unitaire.

Notre contribution a donc consisté en quatre points :

- **L'analyse des modèles de cohérence appliquée à un flot de conception (Chapitre 2).** Cette analyse se base sur une analogie qui établit que le flot de conception des systèmes sur puce peut être vu comme un système distribué, où les nœuds indépendants seraient les équipes (ou les acteurs) du flot, les messages échangés entre les nœuds seraient les documents échangés par les équipes et les opérations effectuées par les nœuds les modifications effectuées sur les documents. Nous avons conclu que le modèle de cohérence initialement appliqué au flot de conception des systèmes sur puce est implicitement le modèle *eventual consistency*. Ce modèle ne garantit pas la cohérence à chaque instant, mais impose que les modifications soient propagées au bout d'un certain temps non spécifié, potentiellement infini.
- **Un moyen d'expliciter les liens de cohérence inter-documentaire initialement implicites (Chapitre 4).** Afin de définir complètement les liens de cohérence entre les documents, nous avons présenté une extrapolation des transformations de modèles bien connues en Ingénierie des Modèles, établissant que toute modification ou création de document peut être considérée comme une transformation, fut elle faite par un humain ou par un programme. Cette extrapolation nous permet d'utiliser des travaux sur la caractérisation des transformations de modèles pour décrire complètement les liens de de cohérence. La formalisation des liens s'appuie sur des extensions du standard de description d'architecture ISO/IEC/IEEE 42010. Ce standard définit les concepts fondamentaux permettant de construire une description d'architecture. Cependant, il offre une base devant être adaptée à chaque cas d'utilisation. Nos extensions permettent de spécialiser les concepts de *view*, d'*architecture model*, et de *correspondence* afin de représenter les liens entre les documents et d'apporter au standard la notion de dynamisme propre au développement d'un système.
- **Une méthodologie d'exploitation des liens afin de garantir un modèle de cohérence explicite dans le flot de conception des systèmes sur puce (Chapitre 4).** L'exploitation des liens permet, d'une part, d'aider les acteurs du flot à propager les modifications intervenues sur un document de façon systématique. D'autre part, notre approche permet de faire émerger le processus réel extrait de l'observation des pratiques des acteurs. En effet, la création et l'enrichissement de la description d'architecture du flot de conception permet de faire ressortir une cartographie des différents documents, en exhibant leur utilisation tant au niveau temporel (à quelles étapes du flot) que spatial (par quelles équipes). De plus, l'application de l'approche permet d'imposer explicitement un modèle de cohérence à mi-chemin entre le modèle *Eager Release Consistency* et *Write-follows-Read Consistency*. En effet, nous

sommes capables de cibler les documents potentiellement impactés par une modification (suivant ainsi le ciblage de l'*Eager Release Consistency*) sans toutefois sacrifier la disponibilité des documents par le système de synchronisation. L'approche supporte l'existence d'incohérences éphémères (*write-follows-read consistency)* jusqu'à ce que les acteurs propagent les modifications après avoir été notifiés.

- **Le développement d'un prototype mettant en œuvre les concepts décrient par notre approche (Chapitre 5).** Ce prototype développé en Java a pour but de tester la validité et l'efficacité de notre approche. Celui-ci se décompose en trois parties : un crochet situé sur le dépôt de version permettant d'intercepter une publication pour en extraire les méta-données, un serveur permettant de stocker les descriptions d'architecture et de l'enrichir grâce aux méta-données transmises par le crochet, et un client sur le poste local des acteurs permettant la communication entre l'acteur et le serveur.

Nous avons évalué l'approche sur un cas d'utilisation concret, extrait des pratiques de STMicroelectronics (Chapitre 6). Cette évaluation, laisse apparaitre que l'utilisation de l'approche permet aux acteurs de corriger 5,5% d'incohérence en plus en un temps moyen de 3,3% plus court. L'approche apporte également un gain indirect, en prévenant l'installation d'incohérences implicites. Ceci permet de faire gagner potentiellement plusieurs semaines (voire des mois) de débogage, raccourcissant ainsi le temps de développement d'un produit.

Cette évaluation ne mesure ni le coût de mise en œuvre de l'approche (l'effort initial pour créer la description d'architecture), ni le retour sur investissement. Il est évident que la définition de la description d'architecture ainsi que la maintenance de la cohérence tout au long du cycle de développement a un coût non nul, en termes de temps de développement. Cependant, il est à noter que la méthodologie entraine la fragmentation de ce coût. La mise en œuvre initiale et son maintien ne sont pas dédiés à une seule personne ou une seule équipe ne faisant que cette tâche. Chaque acteur est responsable de ses fragments de description d'architecture et se leurs maintiens. De plus, le retour sur investissement est une notion très compliquée à mesurer dans un cadre industriel. En effet, mesurer ce retour sur investissement sur un projet réel nécessiterait de réaliser ce projet deux fois, une avec et une sans l'approche. Cependant, il n'est pas possible dans une industrie de monopoliser des ressources sur plusieurs semaines, voire plusieurs mois, pour évaluer une solution.

Les travaux devraient être prolongés afin de compléter le prototype dans le but de le rendre complètement exploitable en production. En effet, le prototype est actuellement dédié à une utilisation avec Subversion. Toutefois l'approche proposée dans ce manuscrit est générique et pourra aisément être portée sur d'autres types de dépôts. Deux options sont possibles pour en assurer le déploiement :

- développer des adaptateurs permettant de récupérer les méta-informations sur chaque logiciel de gestion de versions (Git, Mercurial etc…).

- trouver un nouveau moyen de récupérer ces méta-informations qui ne se baserait plus sur le logiciel en question mais sur l'envoi de données par exemple.

Dans le futur, l'automatisation de la solution pourra être complétée. En effet, le but ultime serait d'automatiser au maximum la propagation des changements dans le flot de conception des systèmes sur puce. L'adoption de l'IDM contribuera à cette automatisation. L'approche proposée ici permettra de systématiser et d'intégrer les flots automatisés.

7.2 Ouvertures

Outre l'implémentation des parties manquantes dont nous venons de parler, l'utilisation d'APPE et de notre approche laisse entrevoir différentes pistes pour des travaux futurs. Nous allons donc, pour conclure ce manuscrit, en énoncer quelques-unes.

Bien que la granularité des fragments liables que nous suggérons soit grosse, la création des liens avec l'approche reste tout de même un travail important même si l'étape de création est répartie entre tous les acteurs. Des milliers de documents sont utilisés afin de concevoir un système sur puce. Maintenir cette base de liens représente un travail titanesque. Il pourrait alors être utile d'investiguer des heuristiques et des algorithmes d'apprentissage automatique permettant de créer automatiquement les liens selon des traitements statistiques effectués sur les publications ou en se rapprochant des méthodes de *data mining* et de *big data*.

Par exemple, une heuristique simple à mettre en œuvre serait la création de lien de co-modification. Lorsqu'un utilisateur prend l'habitude de modifier en même temps plusieurs documents, nous pourrions en déduire que ces documents sont liés entre eux, et alors notifier l'utilisateur lorsque celui-ci modifie un des documents sans modifier les autres. L'implémentation de cette heuristique pourrait se faire en suivant l'algorithme de la colonie de fourmis. Chaque fois que l'utilisateur publie des modifications sur un ensemble de documents, une trace est créée ou renforcée entre les documents jusqu'à atteindre un certain seuil où la trace deviendrait un lien à part entière.

Notre approche est adaptable à tous niveaux de granularité, bien que pour nos premières expérimentations nous ayons choisi de mettre à disposition la granularité au niveau document. Nous avons, dans ce manuscrit, évoqué les avantages et inconvénients du grain fin et du gros grain. Le choix ou la définition d'un grain optimum ne peut, cependant, pas être défini dans l'instant. En effet, il conviendrait de proposer une approche adaptant automatiquement le grain des fragments liables en se servant des mécanismes d'observations des pratiques des acteurs que nous avons mis en place.

Notre approche se veut transverse dans le flot de conception des systèmes sur puce. Cependant, l'implémentation que nous en avons faite la laisse indépendante des différents outils et langages utilisés. Il conviendrait de modifier notre prototype pour que celui-ci implémente les spécification et recommandations de la communauté OSLC. Ceci aurait pour

objectif de rendre l'approche, non plus transverse, mais centrale en lui permettant d'une part d'exploiter des fragments fournis par les différents outils, mais également de partager des fragments de la description d'architecture avec d'autres outils pour maximiser l'automatisation de la maintenance de cohérence. L'effort nécessaire pour implémenter une solution s'accordant avec les spécifications OSLC porterait surtout sur la réflexion de la fragmentation de la description d'architecture.

Enfin, l'émergence de processus présentée dans ce manuscrit permettrait à terme de contrôler et de juger de la validité de modifications apportées au flot usuelle macroscopique. Ainsi, les déviances de pratiques exercées par certains acteurs pourraient soit être acceptées et intégrées dans les pratiques encouragées soit refusées. Cependant, cette possible utilisation de nos travaux, nous pousse à chercher une nouvelle définition de la cohérence d'un corpus documentaire. En effet, la définition que nous proposons dans ce manuscrit est marquée par un caractère manichéen prononcé : le corpus documentaire est cohérent ou incohérent. Cette définition laisse entendre qu'aucune amélioration ne peut être apportée. Cependant, l'évaluation de l'apport d'une nouvelle pratique doit pouvoir être jugée et nécessite donc la possibilité de quantifier la cohérence plus finement.

Bibliographie

[1] L. Gouzenes, “Des composants électroniques toujours plus petits et performants,” in *Annales des Mines-Réalités industrielles*, 2009, pp. 32–35.

[2] “ISO/IEC/IEEE Systems and software engineering – Architecture description,” *ISOIECIEEE 420102011E Revis. ISOIEC 420102007 IEEE Std 1471-2000*, pp. 1–46, Dec. 2011.

[3] Aurélien Chichignoud, Florian Noyrit, Laurent Maillet-Contoz, and François Terrier, “Use of architecture description to maintain consistency in agile processes,” in *Use of architecture description to maintain consistency in agile processes*, 2017.

[4] J. Bézivin, R. F. Paige, U. A. smann, B. Rumpe, and D. Schmidt, “Manifesto - Model Engineering for Complex Systems,” *CoRR*, vol. abs/1409.6591, 2014.

[5] “Advanced FTA STB processor with integrated DVB-C/DVB-T/DVB-T2 demodulator.” [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/data_brief/84/4b/4a/b0/ff/eb/41/3a/DM00087574.pdf/files/DM00087574.pdf/jcr:content/translations/en.DM00087574.pdf.

[6] “Les systèmes sur puce de STMicroelectronics équipent les nouveaux décodeurs satellite Sky Q.” [Online]. Available: http://www.st.com/content/st_com/en/about/media-center/press-item.html/fr/t3772.html. [Accessed: 03-Nov-2016].

[7] *ST231 Core and Instruction Set Architecture Manual*. 2005.

[8] L. Maillet-Contoz and F. Ghenassia, “Transaction level modeling,” in *Transaction Level Modeling with SystemC*, Springer, 2005, pp. 23–55.

[9] D. E. Thomas, E. D. Lagnese, R. A. Walker, J. V. Rajan, R. L. Blackburn, and J. A. Nestor, *Algorithmic and Register-Transfer Level Synthesis: The System Architect’s Workbench: The System Architect’s Workbench*, vol. 85. Springer Science & Business Media, 1989.

[10] I. MathWorks, *MATLAB: the language of technical computing. Desktop tools and development environment, version 7*, vol. 9. MathWorks, 2005.

[11] R. Z. ITU-T and Z. Recommendation, “100: Specification and Description Language (SDL),” *Int. Telecommun. Union*, 2000.

[12] G. Berry and G. Gonthier, “The Esterel synchronous programming language: Design, semantics, implementation,” *Sci. Comput. Program.*, vol. 19, no. 2, pp. 87–152, 1992.

[13] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[14] A. S. Vincentelli and J. Cohn, “Platform-based design,” *IEEE Des. E Test*, vol. 18, no. 6, pp. 23–33, 2001.

[15] R. C. Leachman and S. Ding, “Integration of speed economics into decision-making for manufacturing management,” *Int. J. Prod. Econ.*, vol. 107, no. 1, pp. 39–55, 2007.

[16] L. Maillet-Contoz and M. Moy, “Prototypage virtuel de système sur puce pour une simulation rapide et fidèle.” [Online]. Available: https://www.college-de-france.fr/media/gerard-berry/UPL8152740393975155926_204_01_29_MailletContoz_seminaireCdF_20140129_vf.pdf.

[17] “IP-XACT.” [Online]. Available: http://www.accellera.org/activities/working-groups/ip-xact. [Accessed: 06-Nov-2015].

[18] E. Michinov, “La distance physique et ses effets dans les équipes de travail distribuées: une analyse psychosociale,” *Trav. Hum.*, vol. 71, no. 1, pp. 1–21, 2008.

[19] E. W. Dijkstra, “On the role of scientific thought,” in *Selected Writings on Computing: A Personal Perspective*, Springer, 1982, pp. 60–66.

[20] K. Forsberg and H. Mooz, “The relationship of system engineering to the project cycle,” in *INCOSE International Symposium*, 1991, vol. 1, pp. 57–65.

[21] T. E. Bell and T. A. Thayer, “Software requirements: Are they really a problem?,” in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 61–68.

[22] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, and others, “Manifesto for agile software development,” 2001.

[23] A. Ghazarian, “Reliability in Agile Software Engineering: A Dilemma,” Annual Technical Report, IEEE, 2010.

[24] P. Svejvig and A.-D. F. er Nielsen, “The dilemma of high level planning in distributed agile software projects: an action research study in a danish bank,” in *Agility Across Time and Space*, Springer, 2010, pp. 171–182.

[25] D. Hoyle, “ISO 9000: quality systems handbook,” 2001.

[26] M. Mosbah, “Modèles et approches formels pour les systèmes distribués.” [Online]. Available: http://www.labri.fr/perso/mosbah/Enseignement/ALGODIST/cours1.pdf.

[27] D. Mosberger, “Memory consistency models,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 27, no. 1, pp. 18–26, 1993.

[28] S. V. Adve and K. Gharachorloo, “Shared memory consistency models: A tutorial,” *computer*, vol. 29, no. 12, pp. 66–76, 1996.

[29] J. Lechtenbörger, “Two-Phase Commit Protocol,” in *Encyclopedia of Database Systems*, Springer, 2009, pp. 3209–3213.

[30] J. B. Carter, J. K. Bennett, and W. Zwaenepoel, *Implementation and performance of Munin*, vol. 25. ACM, 1991.

[31] E. A. Brewer, “Towards robust distributed systems,” in *PODC*, 2000, vol. 7.

[32] D. J. Abadi, “Consistency tradeoffs in modern distributed database system design,” *Comput.-IEEE Comput. Mag.*, vol. 45, no. 2, p. 37, 2012.

[33] V. Hilderman and T. Baghi, *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)*. Avionics Communications, 2007.

[34] R. Palin, D. Ward, I. Habli, and R. Rivett, “ISO 26262 safety cases: compliance and assurance,” in *System Safety, 2011 6th IET International Conference on*, 2011, pp. 1–6.

[35] D. Systèmes, *Reqtify—Dassault Systèmes*. www.reqtify.com/, 2013.

[36] R. Avanthi and P. Sreenivasan, “Managing requirements across analysis and design phases using IBM rational system architect & IBM rational DOORS,” Technical report, IBM, 2010.

[37] O. C. Gotel and A. C. Finkelstein, “An analysis of the requirements traceability problem,” in *Requirements Engineering, 1994., Proceedings of the First International Conference on*, 1994, pp. 94–101.

[38] D. Geer, “Eclipse becomes the dominant Java IDE,” *Computer*, vol. 38, no. 7, pp. 16–18, 2005.

[39] “Eclipse.org.” [Online]. Available: http://eclipse.org/.

[40] S. Guckenheimer and J. J. Perez, *Software Engineering with Microsoft Visual Studio Team System (Microsoft. NET Development Series)*. Addison-Wesley Professional, 2006.

[41] Y. Kashai, “SoC Block-Based Design and IP Assembly,” *Electron. Des. Autom. IC Syst. Des. Verification Test.*, p. 75, 2016.

[42] "Magillem Content Platform." [Online]. Available: http://www.magillem.com/wp-content/uploads/2016/01/3V_MCP_web.pdf.

[43] C. Hein, T. Ritter, and M. Wagner, "Model-driven tool integration with modelbus," in *Workshop Future Trends of Model-Driven Development*, 2009, pp. 50–52.

[44] "Modelbus Overview." [Online]. Available: https://www.modelbus.org/en/modelbusoverview.html.

[45] O. C. S. Workgroup, "OSLC core specification version 2.0," *Open Serv. Lifecycle Collab. Tech Rep*, 2010.

[46] E. Architect, *Sparx Systems*. Inc, 2010.

[47] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.

[48] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, "Using description logic to maintain consistency between UML models," in *«UML» 2003-The Unified Modeling Language. Modeling Languages and Applications*, Springer, 2003, pp. 326–340.

[49] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelsteiin, "xlinkit: A consistency checking and smart link generation service," *ACM Trans. Internet Technol. TOIT*, vol. 2, no. 2, pp. 151–185, 2002.

[50] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *Softw. Eng. IEEE Trans. On*, vol. 37, no. 2, pp. 188–204, 2011.

[51] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, 2008, pp. 511–520.

[52] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Inf. Syst. J.*, vol. 20, no. 5, pp. 449–480, 2010.

[53] C. R. Prause and Z. Durdik, "Architectural design and documentation: Waste in agile development?," in *Software and System Process (ICSSP), 2012 International Conference on*, 2012, pp. 130–134.

[54] "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," *IEEE Std 1471-2000*, p. i-23, 2000.

[55] "ISO and IEEE publish new edition of standard for architecture description of systems." [Online]. Available: http://www.iso-architecture.org/42010/pr-42010-2011-12.html.

[56] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, "Viewpoints: A framework for integrating multiple perspectives in system development," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 2, no. 1, pp. 31–57, 1992.

[57] P. Clements, M. J. Escalona, P. Inverardi, I. Malavolta, and E. Marchetti, "Exploiting software architecture to support requirements satisfaction testing," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 484–487.

[58] I. Malavolta, "Software Architecture Modeling by Reuse, Composition and Customization," *Univ. L'Aquila L'Aquila Italy Thesis*, vol. 1, 2012.

[59] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[60] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol. TOSEM*, vol. 6, no. 3, pp. 213–249, 1997.

[61] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," DTIC Document, 2006.

[62] J. Li, N. T. Pilkington, F. Xie, and Q. Liu, "Embedded architecture description language," *J. Syst. Softw.*, vol. 83, no. 2, pp. 235–252, 2010.

[63] V. Debruyne, F. Simonot-Lion, and Y. Trinquet, "East-adl-an architecture description language-validation and verification aspects," in *IFIP Workshop on Architecture Description Languages 2004-WADL'04*, 2004, pp. 53–62.

[64] R. Van Ommering, F. Van Der Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," *Computer*, vol. 33, no. 3, pp. 78–85, 2000.

[65] F. Oquendo, "π-ADL: an Architecture Description Language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures," *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 3, pp. 1–14, 2004.

[66] E. Bruneton, T. Coupaye, and J.-B. Stefani, "The fractal component model," *Draft Specif. Version*, vol. 2, no. 3, 2004.

[67] M. S. Mohammad, "A formal component-based software engineering approach for developing trustworthy systems," Concordia University, 2009.

[68] R. Hilliard and T. Rice, "Expressiveness in architecture description languages," in *Proceedings of the third international workshop on Software architecture*, 1998, pp. 65–68.

[69] R. Pandey, "Architectural description languages (ADLs) vs UML: a review," *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 3, pp. 1–5, 2010.

[70] E. Woods, “Architecture description languages and information systems architects: Never the twain shall meet?,” *Artechra White Pap.*, 2005.

[71] E. Woods and R. Hilliard, “Architecture Description Languages in Practice Session Report.,” in *WICSA*, 2005, vol. 5, pp. 243–246.

[72] C. Hofmeister, R. Nord, and D. Soni, *Applied software architecture*. Addison-Wesley Professional, 2000.

[73] P. B. Kruchten, “The 4+ 1 view model of architecture,” *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, 1995.

[74] D. Garlan and A. J. Kompanek, “Reconciling the needs of architectural description with object-modeling notations,” in *International Conference on the Unified Modeling Language*, 2000, pp. 498–512.

[75] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins, “Modeling software architectures in the Unified Modeling Language,” *ACM Trans. Softw. Eng. Methodol. TOSEM*, vol. 11, no. 1, pp. 2–57, 2002.

[76] J. E. Robbins, N. Medvidovic, D. F. Redmiles, and D. S. Rosenblum, “Integrating architecture description languages with a standard design method,” in *Proceedings of the 20th international conference on Software engineering*, 1998, pp. 209–218.

[77] J. E. Pérez-Martínez and A. Sierra-Alonso, “UML 1.4 versus UML 2.0 as languages to describe software architectures,” in *European Workshop on Software Architecture*, 2004.

[78] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, and J. R. Silva, “Documenting component and connector views with UML 2.0,” DTIC Document, 2004.

[79] S. Roh, K. Kim, and T. Jeon, “Architecture modeling language based on uml2. 0,” in *Software Engineering Conference, 2004. 11th Asia-Pacific*, 2004, pp. 663–669.

[80] J. Aldrich, G. T. Leavens, M. Barnett, N. Sharygina, and D. Giannakopoulou, “Specification and verification of component-based systems 2007,” in *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers*, 2007, pp. 609–610.

[81] R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, “A model-driven approach to automate the propagation of changes among Architecture Description Languages,” *Softw. Syst. Model.*, vol. 11, no. 1, pp. 29–53, 2012.

[82] P. André, “Transformation de modeles,” 2011.

[83] F. Jouault, “Contribution à l’étude des langages de transformation de modèles,” Nantes, 2006.

[84] D. C. Schmidt, “Model-driven engineering,” *Comput.-IEEE Comput. Soc.-*, vol. 39, no. 2, p. 25, 2006.

[85] J. Bézivin and J.-P. Briot, “Sur les principes de base de l’ingénierie des modèles.,” *L’OBJET*, vol. 10, no. 4, pp. 145–157, 2004.

[86] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger, “Bidirectional transformations: A cross-discipline perspective,” in *Theory and Practice of Model Transformations*, Springer, 2009, pp. 260–283.

[87] K. Czarnecki and S. Helsen, “Classification of model transformation approaches,” in *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003, vol. 45, pp. 1–17.

[88] K. Czarnecki and S. Helsen, “Feature-based survey of model transformation approaches,” *IBM Syst. J.*, vol. 45, no. 3, pp. 621–645, 2006.

[89] P. Stevens, “Bidirectional model transformations in QVT: Semantic issues and open questions,” in *Model Driven Engineering Languages and Systems*, Springer, 2007, pp. 1–15.

[90] S. Hidaka, M. Tisi, J. Cabot, and Z. Hu, “Feature-based classification of bidirectional transformation approaches,” *Softw. Syst. Model.*, pp. 1–22, 2015.

[91] C. Dejours, D. Dessors, and P. Molinier, “Comprendre la résistance au changement,” *Doc. Médecin Trav.*, vol. 58, no. 2, pp. 112–117, 1994.

[92] STMicroelectronics, “STM32F427xx STM32F429xx datasheet.” STMicroelectronics.

[93] “DiffDog.” [Online]. Available: https://www.altova.com/fr/diffdog.html.

[94] “WinMerge.” [Online]. Available: http://winmerge.org/.

[95] “EMF Diff/Merge.” [Online]. Available: http://www.eclipse.org/diffmerge/.

[96] A. Abadi, M. Nisenson, and Y. Simionovici, “A traceability technique for specifications,” in *2008 The 16th IEEE International Conference on Program Comprehension*, 2008.

[97] J. I. Maletic, E. V. Munson, A. Marcus, and T. N. Nguyen, “Using a hypertext model for traceability link conformance analysis,” in *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, 2003, pp. 47–54.

[98] H.-Y. Jiang, T. N. Nguyen, X. Chen, H. Jaygarl, and C. K. Chang, “Incremental latent semantic indexing for automatic traceability link evolution management,” in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008, pp. 59–68.

[99] B. P. Model, “Notation (BPMN) Version 2.0,” *OMG Specif. Object Manag. Group*, 2011.

[100] S. A. White, “Introduction to BPMN,” *IBM Coop.*, vol. 2, no. 0, p. 0, 2004.

[101] N. M. A. Munassar and A. Govardhan, “A comparison between five models of software engineering,” *IJCSI*, vol. 5, pp. 95–101, 2010.

[102] S. Balaji and M. S. Murugaiyan, “Waterfall vs. V-Model vs. Agile: A comparative study on SDLC,” *Int. J. Inf. Technol. Bus. Manag.*, vol. 2, no. 1, pp. 26–30, 2012.

[103] B. W. Boehm, “A spiral model of software development and enhancement,” *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[104] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, “How developers drive software evolution,” in *Principles of Software Evolution, Eighth International Workshop on*, 2005, pp. 113–122.

[105] K. Birman and T. Joseph, *Exploiting virtual synchrony in distributed systems*, vol. 21. ACM, 1987.

[106] J. Cleland-Huang, C. K. Chang, and M. Christensen, “Event-based traceability for managing evolutionary change,” *Softw. Eng. IEEE Trans. On*, vol. 29, no. 9, pp. 796–810, 2003.

[107] J. Jeston and J. Nelis, *Business process management*. Routledge, 2014.

[108] M. Riebisch, “Supporting evolutionary development by feature models and traceability links,” in *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, 2004, pp. 370–377.

[109] H. Kagdi, J. I. Maletic, and B. Sharif, “Mining software repositories for traceability links,” in *15th IEEE International Conference on Program Comprehension (ICPC’07)*, 2007, pp. 145–154.

[110] J. Cleland-Huang, O. Gotel, A. Zisman, and others, *Software and systems traceability*, vol. 2. Springer, 2012.

[111] B. Collins-Sussman, B. Fitzpatrick, and M. Pilato, “Implementing repository hooks,” in *Version control with subversion*, O’Reilly Media, Inc., 2004, pp. 147–149.

[112] “Customizing Git - Git Hooks,” *Customizing Git - Git Hooks*. [Online]. Available: https://git-scm.com/book/it/v2/Customizing-Git-Git-Hooks.

[113] “Handling repository events with hooks,” *Handling repository events with hooks*. [Online]. Available: http://hgbook.red-bean.com/read/handling-repository-events-with-hooks.html.

[114] TMate Software, “SVNKit::Subversion for Java.” [Online]. Available: https://svnkit.com/index.html.

[115] E. Koutsofios, S. North, and others, “Drawing graphs with dot,” Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, 1991.

[116] “Embedding Git in your applications - JGit.” [Online]. Available: https://git-scm.com/book/fr/v2/Embedding-Git-in-your-Applications-JGit.

[117] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, “Goal question metric (gqm) approach,” *Encycl. Softw. Eng.*, 2002.

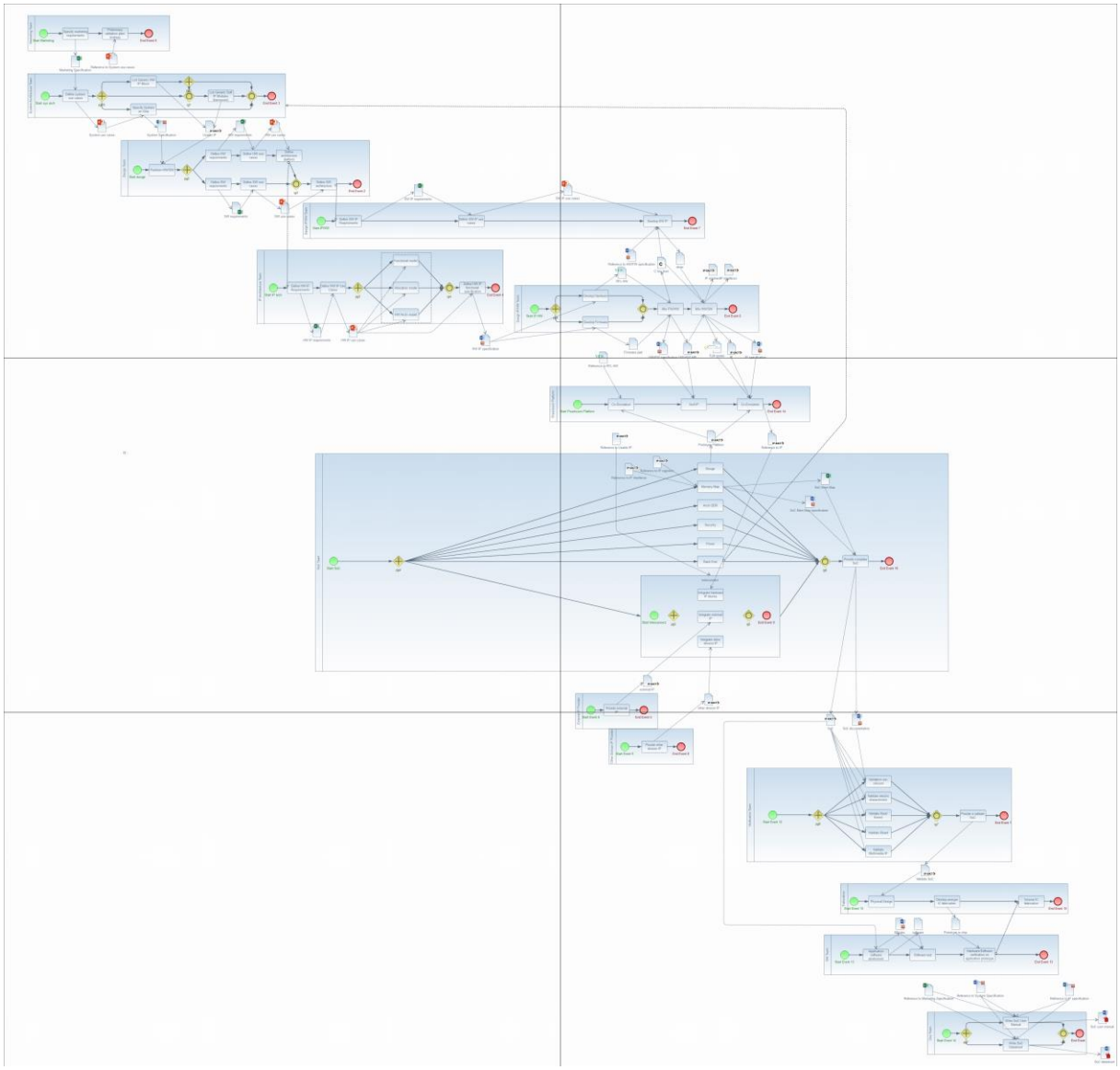
[118] S. S. Shapiro and M. B. Wilk, “An analysis of variance test for normality (complete samples),” *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

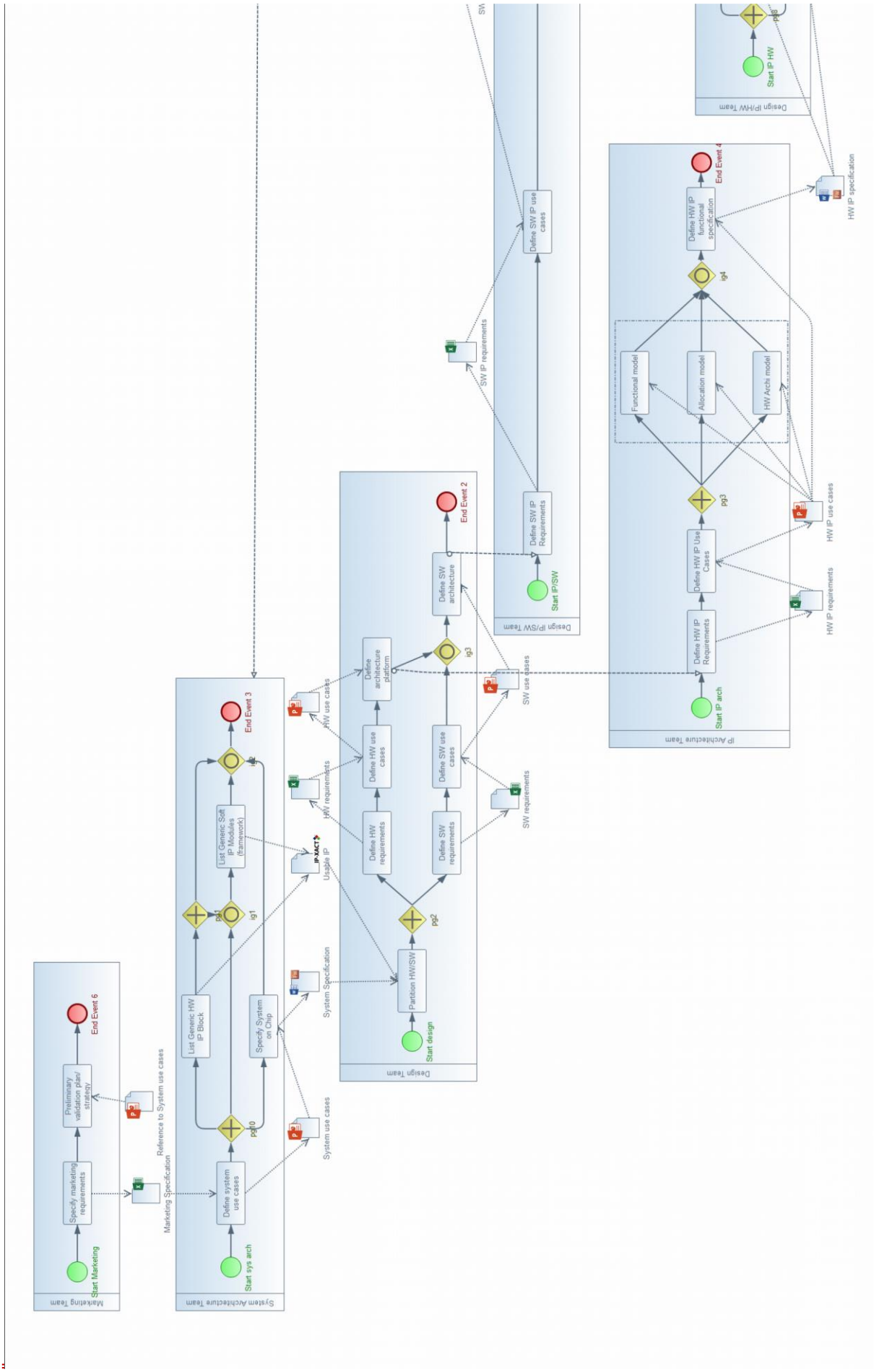
[119] Student, “The probable error of a mean,” *Biometrika*, pp. 1–25, 1908.

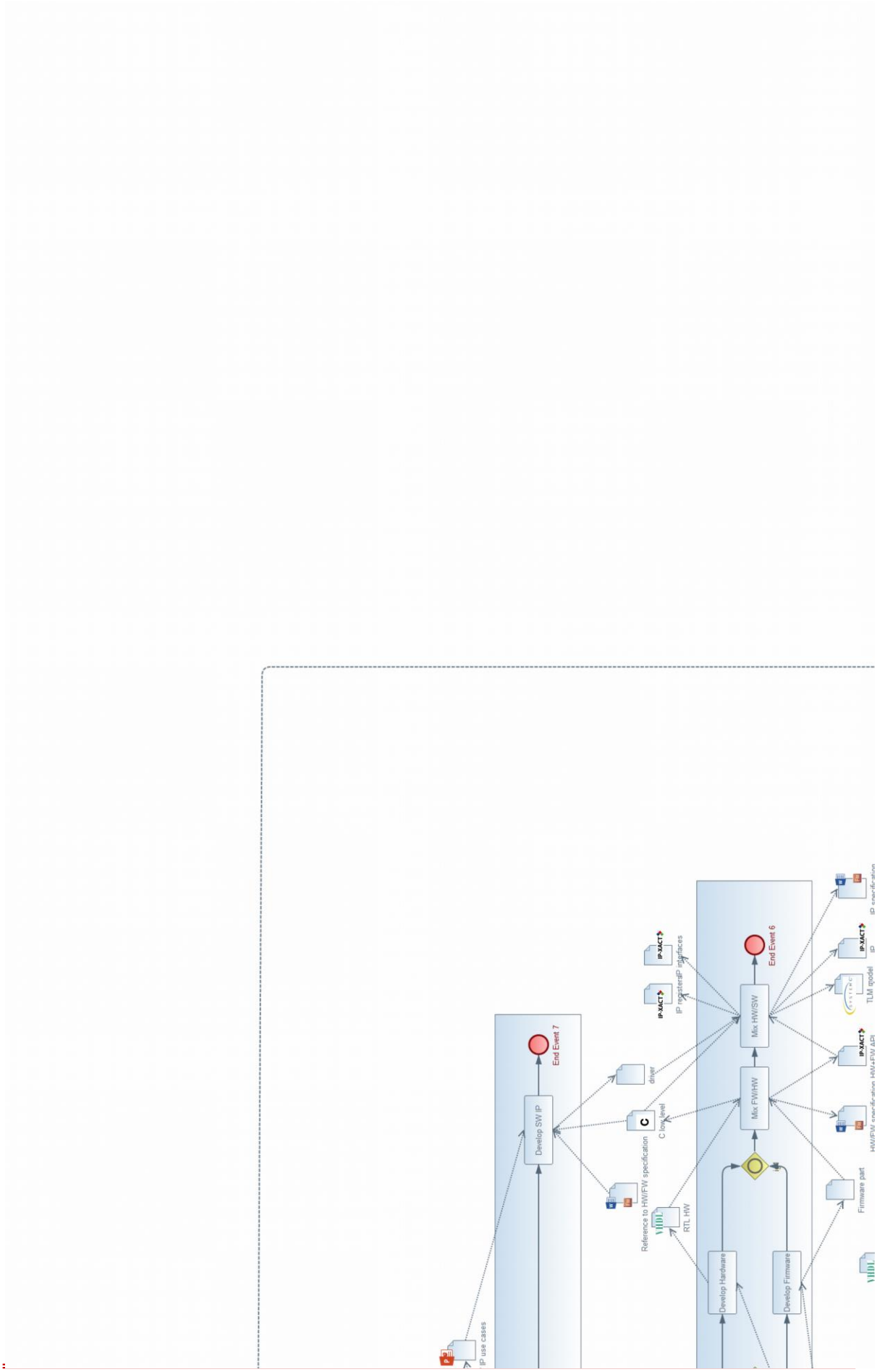
ANNEXES

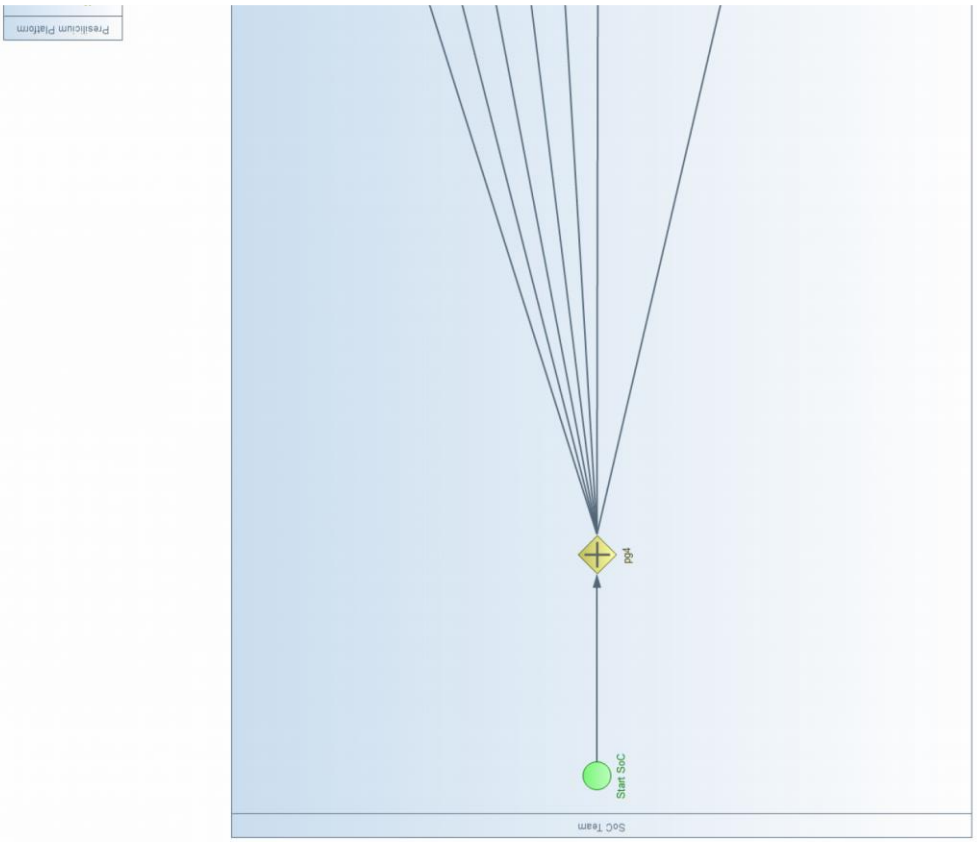
Table des annexes

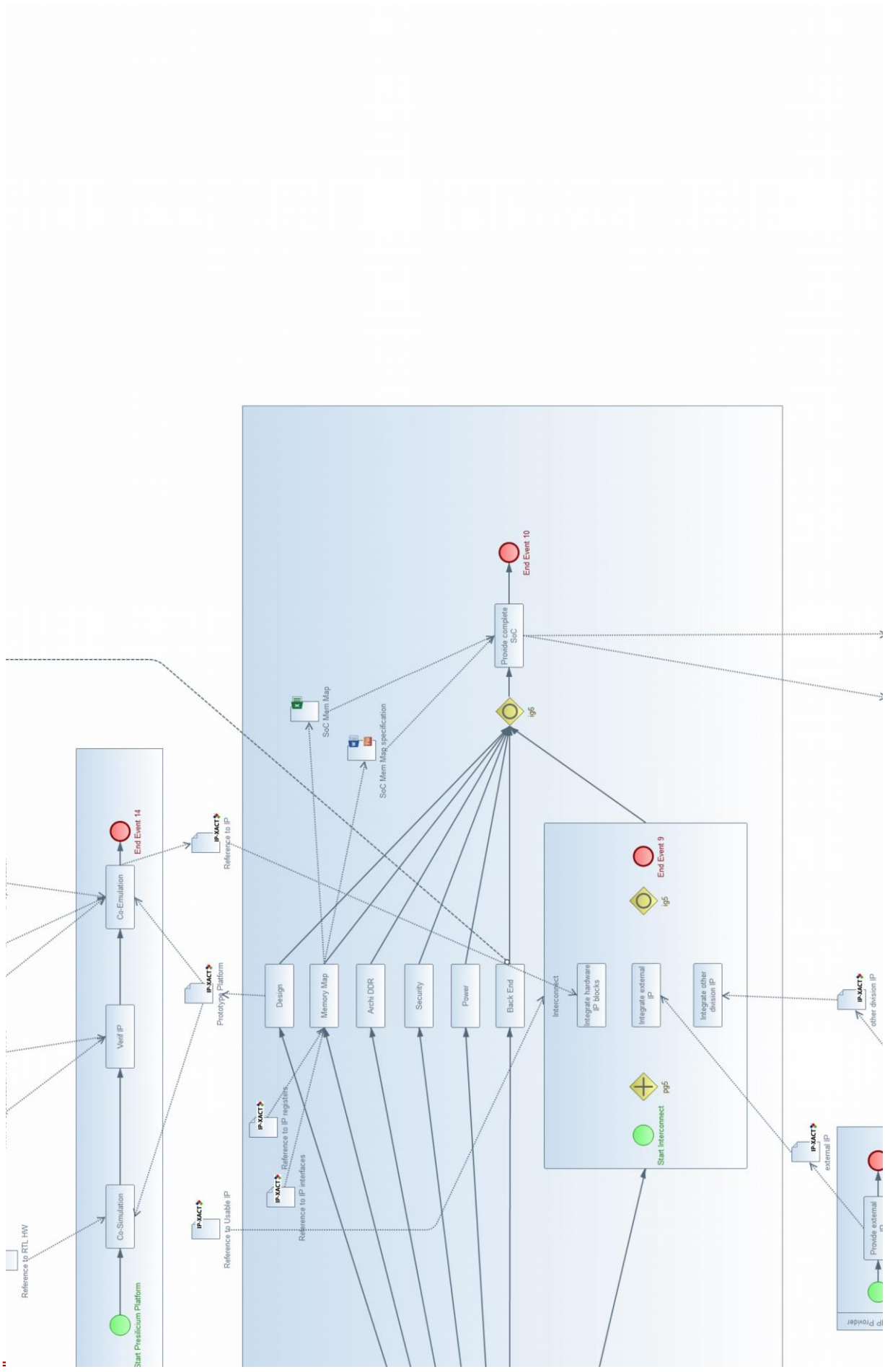
Annexe 1. Processus de conception des systèmes sur puce (BPML)

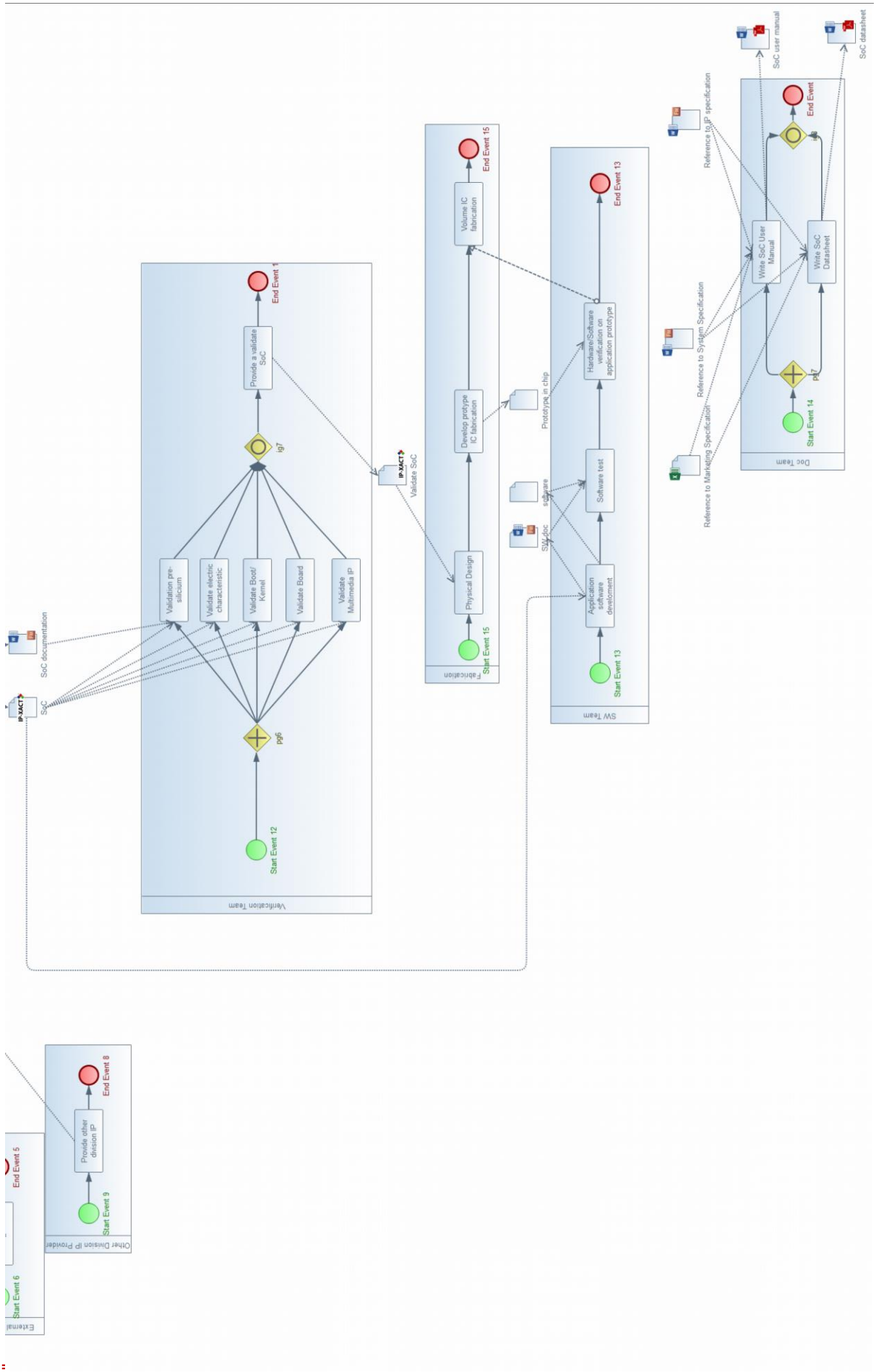



Marketing Team
Start Marketing
Specify marketing requirements
Preliminary validation plan/ strategy
End Event 6
Reference to System use cases
Marketing Specification
System Architecture Team
Start sys arch
Define system use cases
pg0
List Generic HW IP Block
Specify System on Chip
List Generic Soft IP Modules (framework)
ig1
End Event 3
System use cases
System Specification
IP-XACT
Usable IP
Design Team
Start design
Partition HW/SW
pg2
Define HW requirements
Define SW requirements
Define HW use cases
Define SW use cases
Define architecture platform
ig3
End Event 2
HW use cases
HW requirements
SW use cases
SW requirements
Design IP/SW Team
Start IP/SW
Define SW IP Requirements
Define SW IP use cases
SW IP requirements
IP Architecture Team
Start IP arch
Define HW IP Requirements
Define HW IP Use Cases
pg3
HW Archi model
Allocation model
Functional model
ig4
End Event 4
Define HW IP functional specification
HW IP use cases
HW IP requirements
HW IP specification
Design IP/HW Team
Start IP HW


IP use cases
Develop SW IP
End Event 7
Develop Hardware
Develop Firmware
Mix FW/HW
Mix HW/SW
End Event 6
IP-XACT
IP registersIP interfaces
driver
C
C low level
Reference to HW/FW specification
RTL HW
Firmware part
TLM gpodel


Start SoC
SoC Team


Start Presilicium Platform
Co-Simulation
Verif IP
Co-Emulation
End Event 14
Référence to RTL HW
Référence to Usable IP
Référence to IP interfaces
Référence to IP registers
Prototype Platform
Référence to IP
SoC Mem Map
SoC Mem Map specification
Design
Memory Map
Archi DDR
Security
Power
Back End
ig6
Provide complete SoC
End Event 10
Interconnect
Integrate hardware IP blocks
Integrate external IP
Integrate other division IP
pg5
Start Interconnect
ig5
End Event 9
other division IP
external IP


Verification Team
Start Event 12
pg6
Validation pre-silicium
Validate electric characteristic
Validate Boot/ Kernel
Validate Board
Validate Multimedia IP
ig7
Provide a validate SoC
End Event 1
SoC documentation
SoC
IP-XACT
Validate SoC
Fabrication
Start Event 15
Physical Design
Develop prototype IC fabrication
Volume IC fabrication
End Event 15
Prototype in chip
SW Doc
software
SW Team
Start Event 13
Application software development
Software test
Hardware/Software verification on application prototype
End Event 13
Doc Team
Start Event 14
pg7
Write SoC Datasheet
Write SoC User Manual
End Event
Reference to IP specification
Reference to System Specification
Reference to Marketing Specification
SoC user manual
SoC datasheet
Other Division IP Provider
Start Event 9
Provide other division IP
End Event 8
End Event 5
Start Event 6

Annexe 2. Exigences fonctionnelles d'APPE

1. Lors d'une publication

APPE_010	Le programme doit pouvoir intercepter une publication
APPE_011	Le programme doit pouvoir être utilisable avec Subversion
APPE_0112	Optionnel : Le programme doit pouvoir être utilisable avec n'importe quel gestionnaire de version.
APPE_012	Lorsqu'un utilisateur publie un nouveau document, une interface graphique doit s'afficher afin d'obtenir des informations à propos du nouveau document.

2. Informations nécessaires

APPE_020	Les informations saisies par l'utilisateur doivent permettre la création, l'enrichissement ou la modification d'une description d'architecture.
APPE_021	Les informations saisies par l'utilisateur doivent permettre de définir comment un document a été créé (manuellement ou automatiquement)
APPE_022	Dans le cas d'une création de document automatique l'utilisateur peut spécifier quel programme a été utilisé.
APPE_023	Les informations saisies par l'utilisateur doivent permettre d'identifier quels documents contiennent de l'information dupliquée.
APPE_024	L'extraction d'informations de la publication doit définir l'auteur du document.
APPE_025	L'extraction d'informations de la publication doit définir les contributeurs du document.
APPE_026	L'extraction d'informations de la publication doit définir la localisation du document.
APPE_027	Le programme doit associer une date à chaque modification.

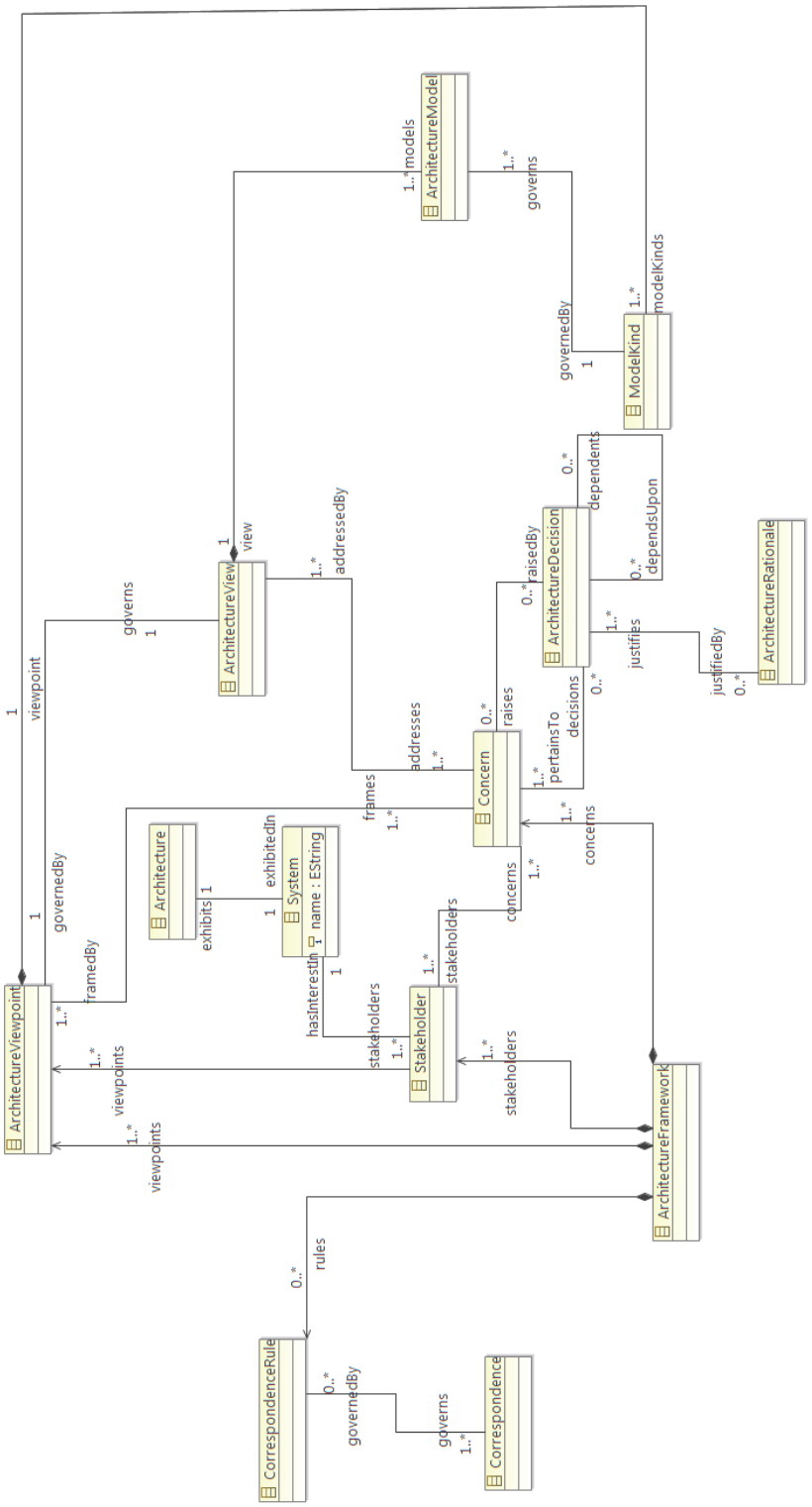
3. Base de données et processus

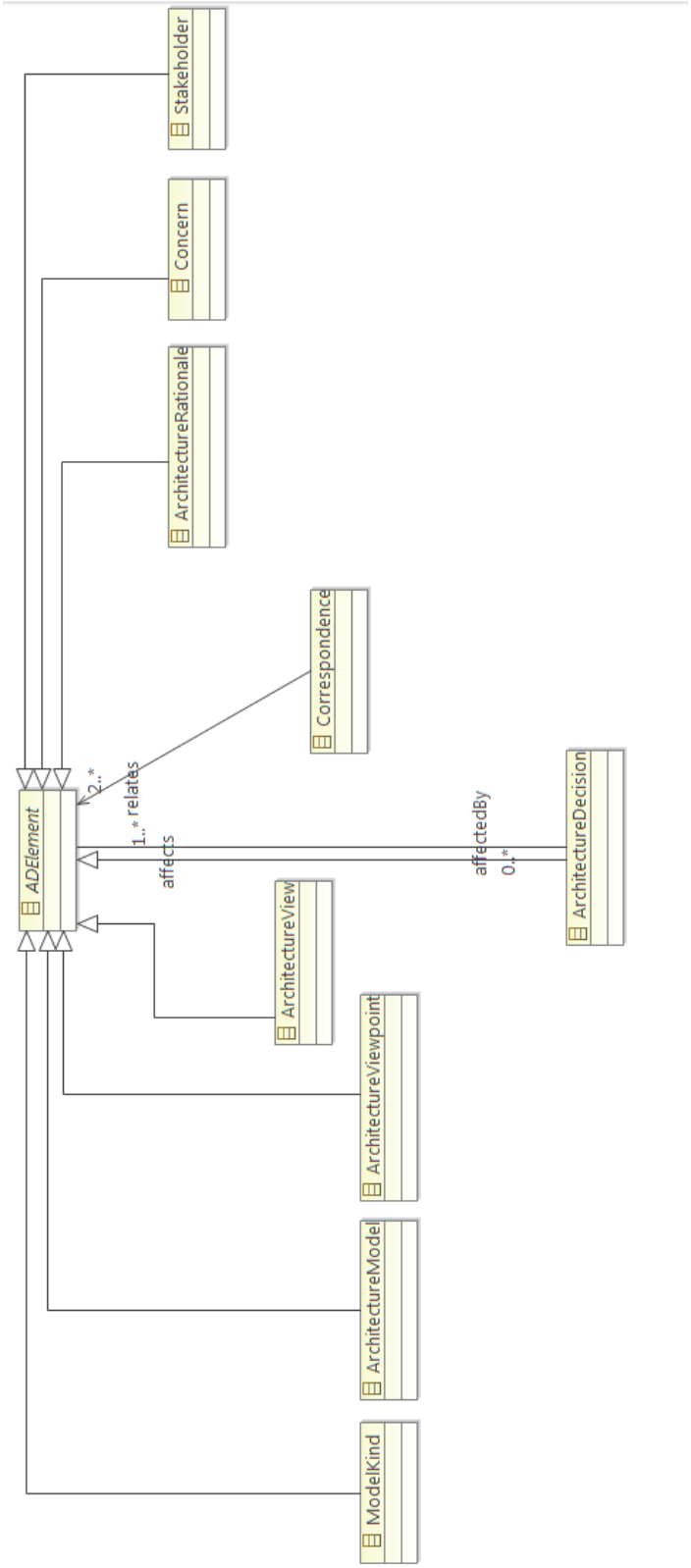
APPE_030	Les informations saisies par l'utilisateur doivent être sauvegardés.
APPE_031	Les informations saisies par l'utilisateur et les informations extraites de la publication doivent pouvoir être partagés par différents utilisateurs.
APPE_032	Les informations saisies par l'utilisateur doivent permettre de visualiser le processus dans une fenêtre graphique.
APPE_0322	Optionnel: Les informations doivent être traitées statistiquement afin de générer automatiquement de nouveaux liens (heuristiques).

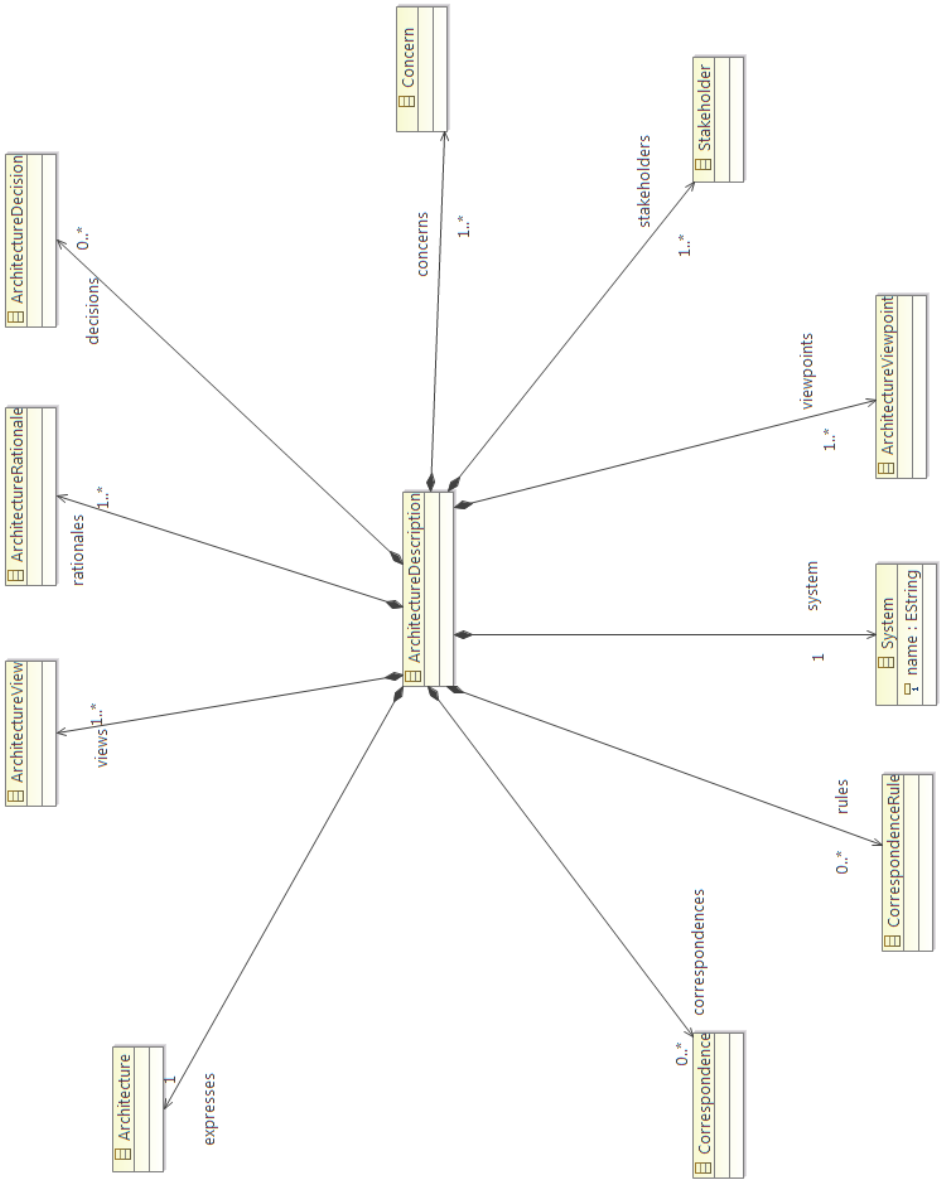
4. Modification d'un document

APPE_040	Le programme doit détecter une modification d'un document.
APPE_041	Le programme doit pouvoir avertir les utilisateurs sur les documents potentiellement impactés par leurs modifications.
APPE_042	Le programme doit avertir les utilisateurs que leurs documents sont potentiellement impactés par une modification.
APPE_043	Le programme doit être capable de lancer la génération de document après approbation de l'utilisateur.

Annexe 3. Diagramme Ecore de description d'architecture




Stakeholder
Concern
ArchitectureRationale
Correspondence
ADElement
2..*
1..* relates
affects
affectedBy
0..*
ArchitectureDecision
ArchitectureView
ArchitectureViewpoint
ArchitectureModel
ModelKind


Concern
Stakeholder
ArchitectureDecision
ArchitectureRationale
ArchitectureView
ArchitectureViewpoint
ArchitectureDescription
System
name : EString
CorrespondenceRule
Architecture
Correspondence
concerns
1..*
stakeholders
1..*
decisions
0..*
rationales
1..*
views
1..*
viewpoints
1..*
system
1
rules
0..*
correspondences
0..*
expresses
1

Annexe 4. Liste exhaustive des fonctions de la classe “CommandExecutor”

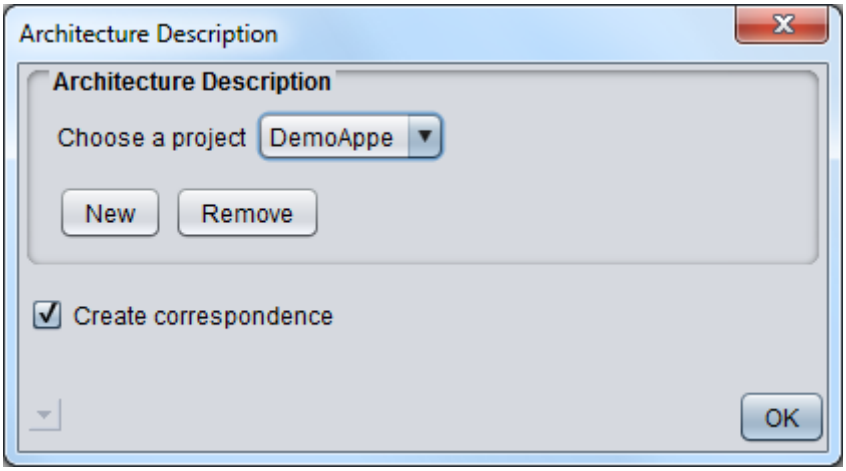
Commande	Argument(s)	Réponse du serveur
actualizeProcess		ACK
addCorrespondenceSource	correspondenceName : String artefactsName : String[]	ACK
commitImpact	commitFiles : String[]	ACK
cancel		ACK
createArchitectureDescription	adName : String	ACK
createArtefacts	stakeholderName : String commitDocument : String[] repository : String transaction : String	Liste des artefacts créés
createCorrespondence	stakeholderName : String sources : String[] targets : String[]	Id de la correspondence
createImpliciteLinks	commitFiles : String[]	ACK
degradeUpdate	artefact : String artefactToRemove : String	ACK
discardNotUpdateArtefact	Artefact : String artefactToRemove : String	ACK
generateDot		Processus en dot
generateDotVersion	artefactName : String	Compatibilité en dot
getAllArtefacts		Liste des artefacts
getAllArtefactsVersions		Liste des ArtefactVersions
getAllCorrespondences		Liste des Correspondences
getArchitectureDescriptions		Liste des ArchitectureDescription
getArtefactVersion	artefactsName : String	Information sur la dernière version d’un Artefact
getCorrespondenceCommand	correspondenceName : String	Information sur la partie *automation* d’une correspondence
getCorrespondences	artefactName : String	Liste de Correspondences impliquant l’Artefact en argument
getCorrespondenceSources	correspondenceName : String	Liste des sources de la Correspondence en argument
getCorrespondenceRelatesVersion	correspondenceName : String	Liste des sources et des cibles de la Correspondence en argument

getImpactedArtefacts	artefactName : String	Liste des Artefacts potentiellement impactés par une modification sur l’Artefact en argument
getModifiedArtefacts	artefactName : String	Information sur l’état de l’Artefact en argument
getNotUpdatedArtefacts	shName : String	Liste des Artefacts potentiellement non à jour relatif au Stakeholder en argument
getStakeholderArtefacts	shName : String	Liste des artefacts relatifs au Stakeholder en argument
getStakeholders		Liste des Stakeholders
informStakeholders	artefactsName : String	ACK
notImpacted	artefactName : String artefactToRemove : String	ACK
removeArchitectureDescription	adName : String	Liste des ArchitectureDescription
removeArtefact	artefactName : String	ACK
removeCorrespondence	correspondenceName : String	Liste des Stakeholders et des Artefacts impliqués dans la Correspondence en argument
removeCorrespondenceSource	shName : String correspondenceName : String source : String	ACK
removeCorrespondenceTarget	shName : String correspondenceName : String target : String	ACK
sendReturnValue		disconnect
setArchitectureDescriptionName	adName : String	ACK
setCorrespondenceCommand	correspondenceName : String automationName : String	ACK

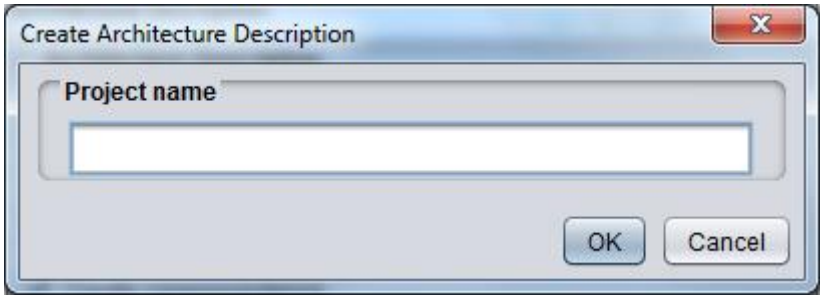
Annexe 5. Interface graphique d'APPE

5. Paquetage commitcatch

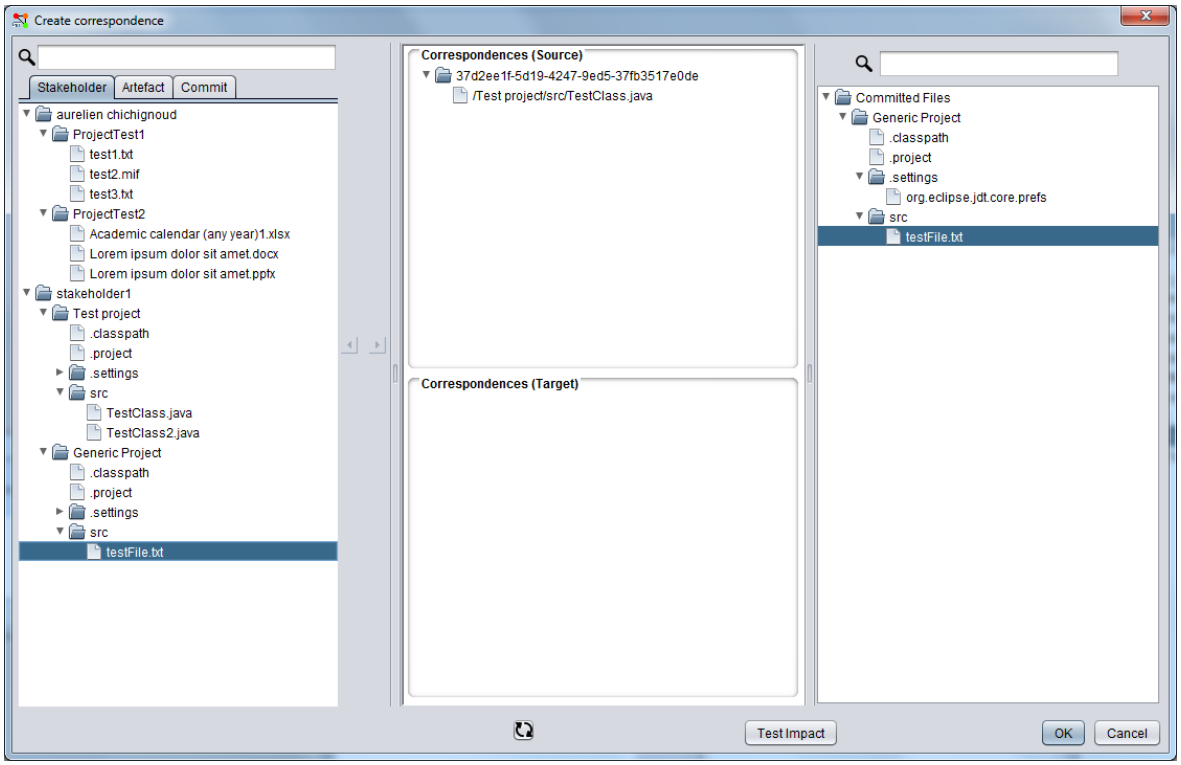
a. DialogPreCorrespondence



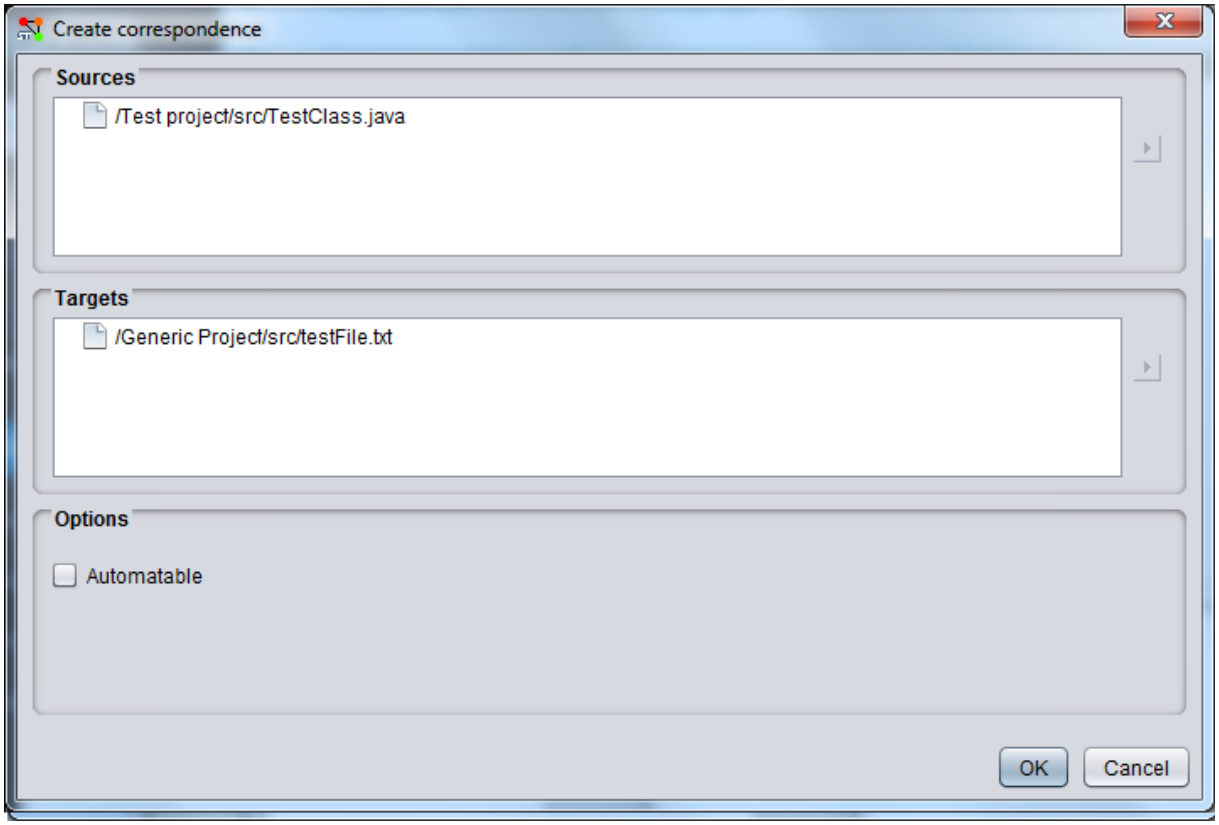
b. WindowADCreation



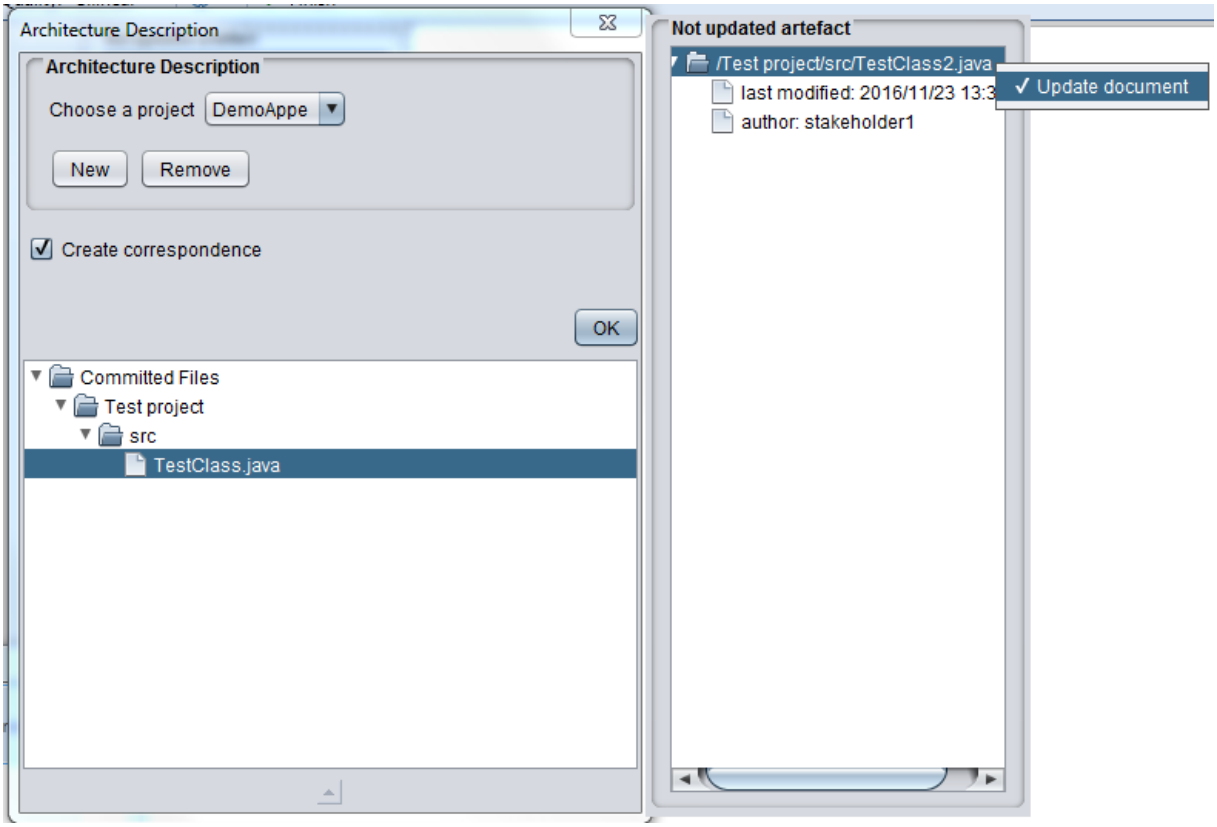
c. WindowCorrespondence



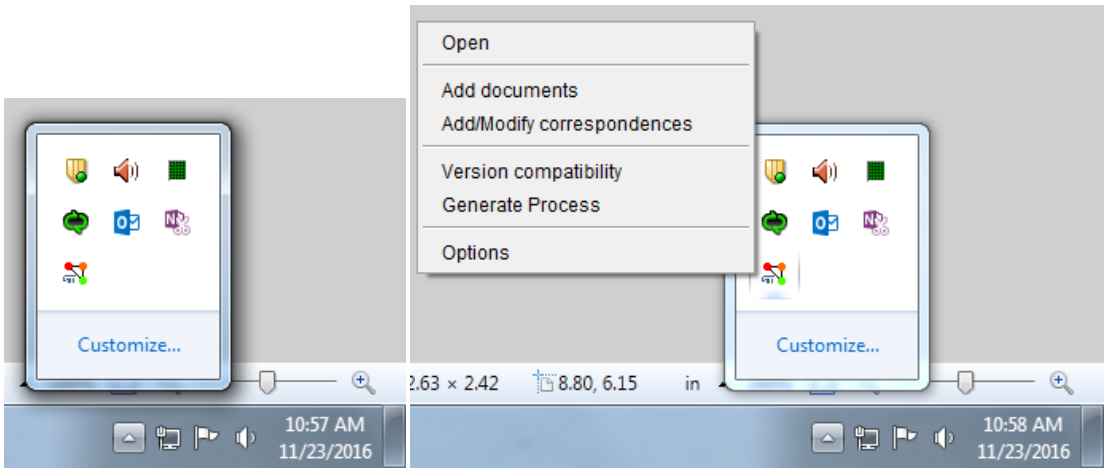
d. DialogCreateCorrespondence



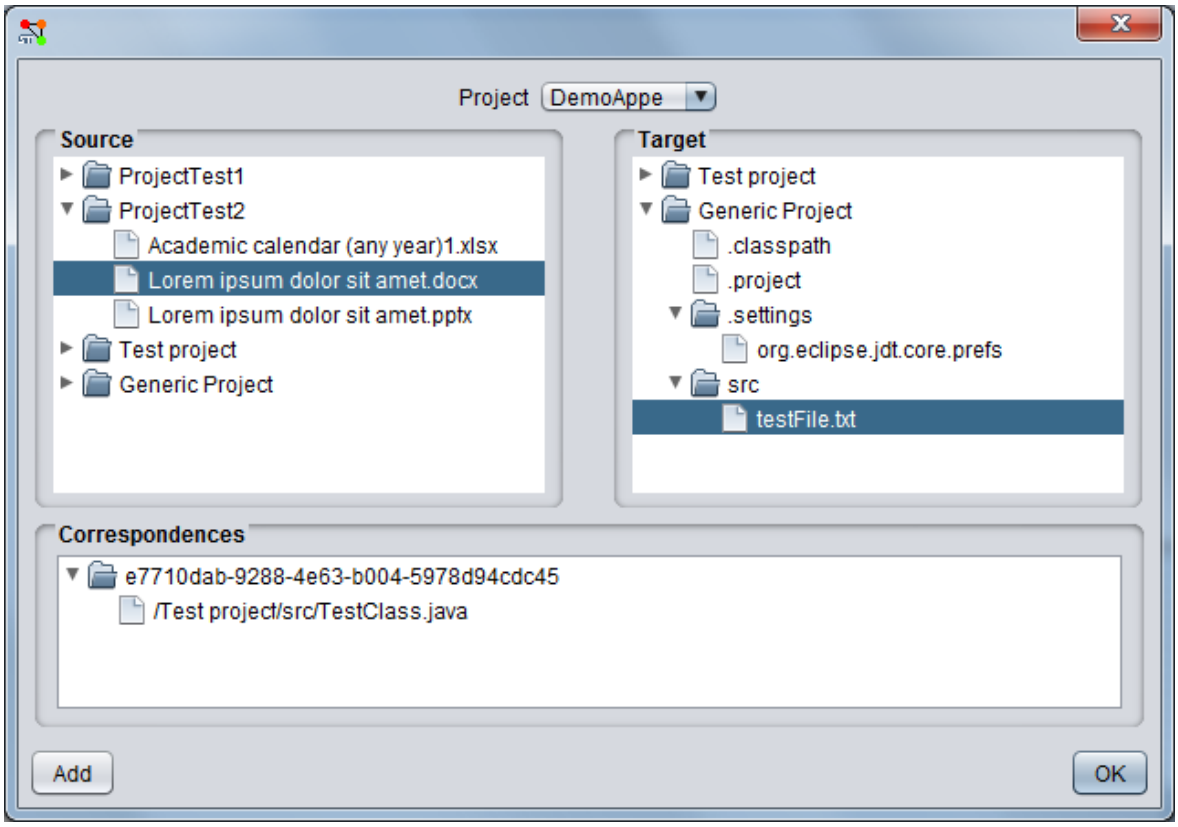
e. DialogNotUpdated



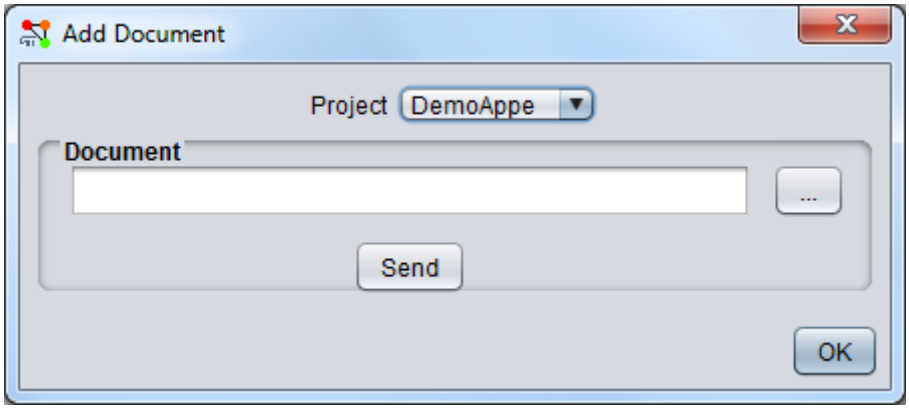
6. Paquetage systemtray



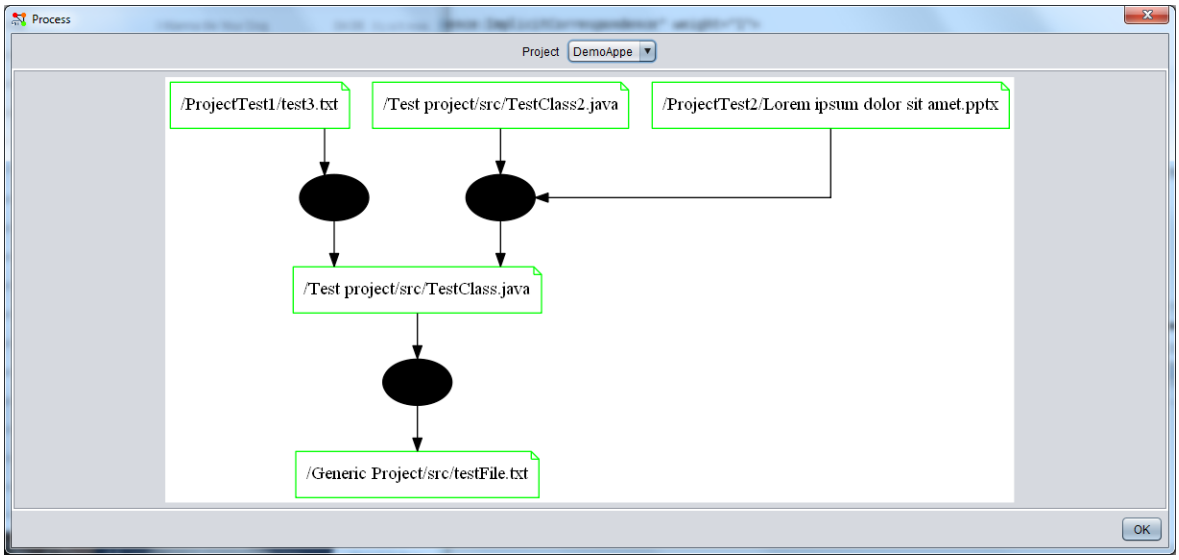
a. DialogAddCorrespondence

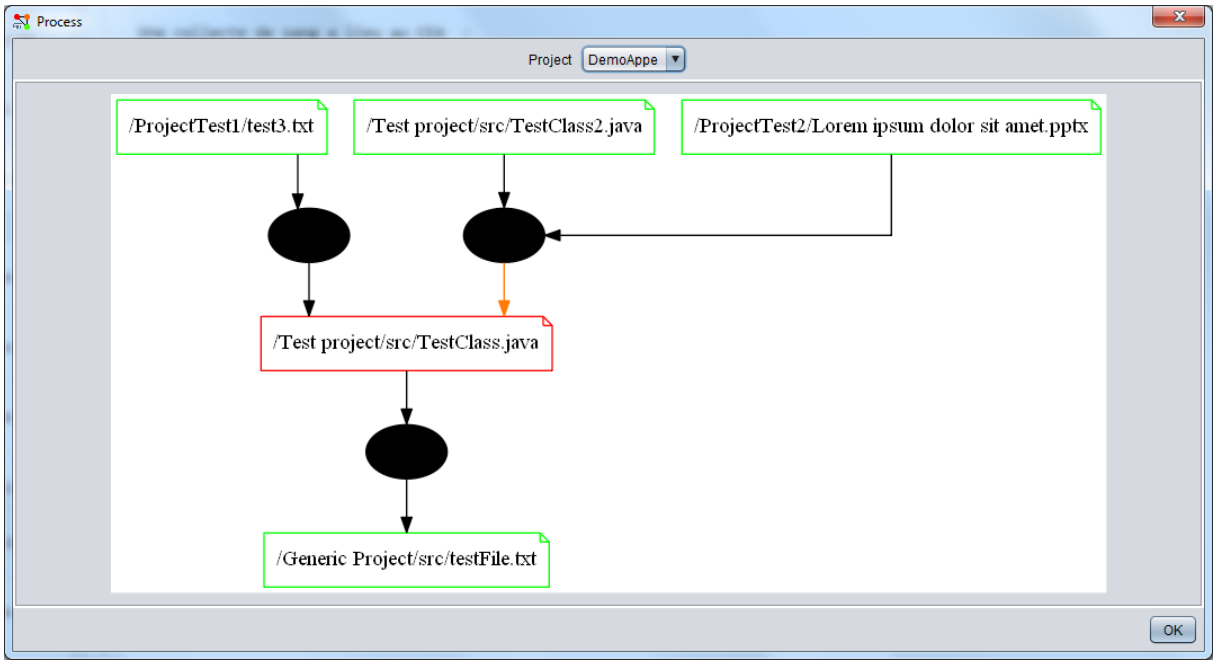


b. DialogAddDocument

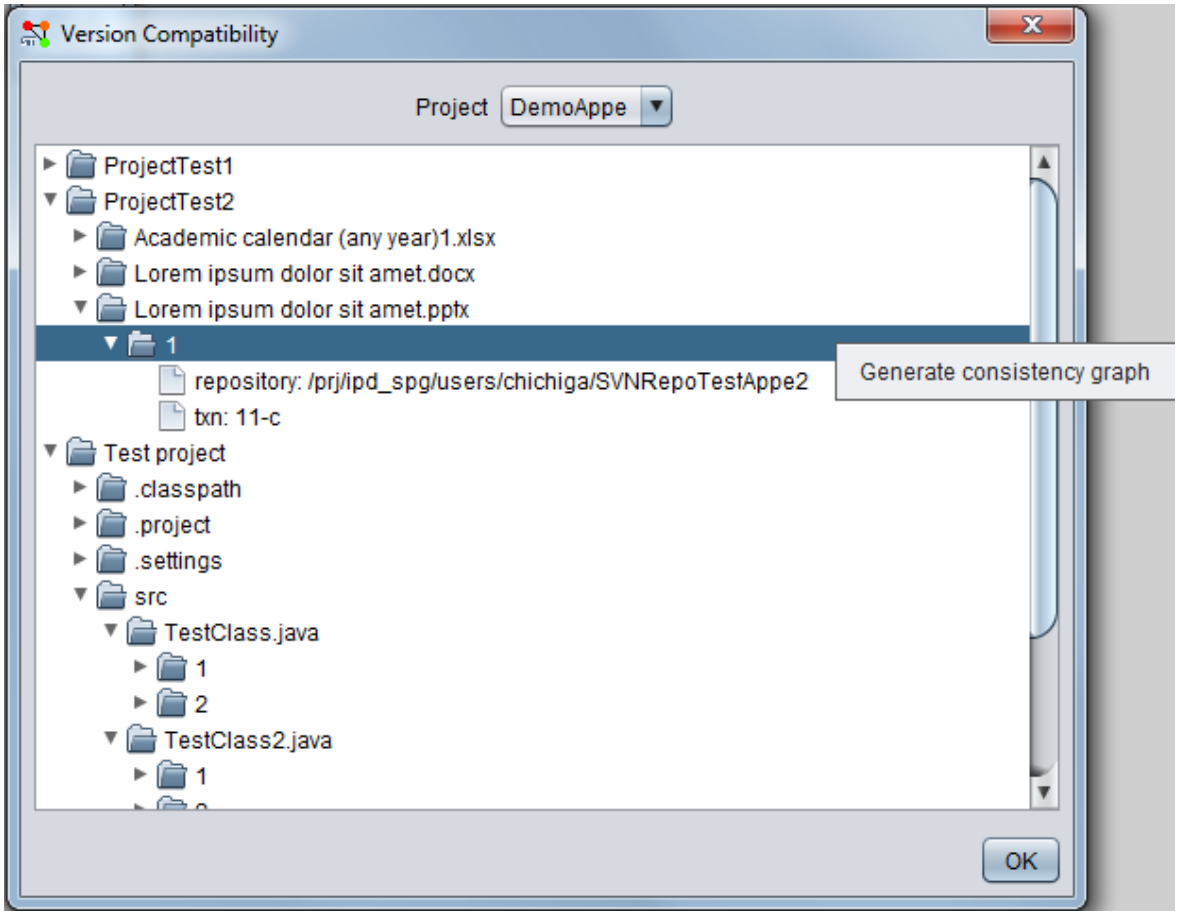


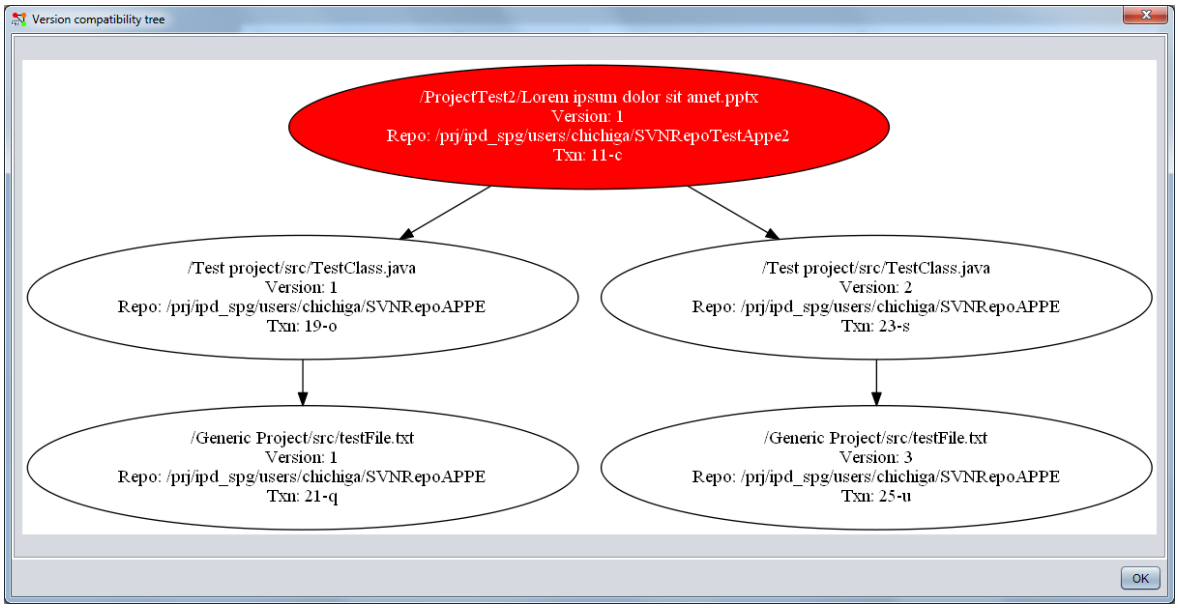
c. DialogProcess



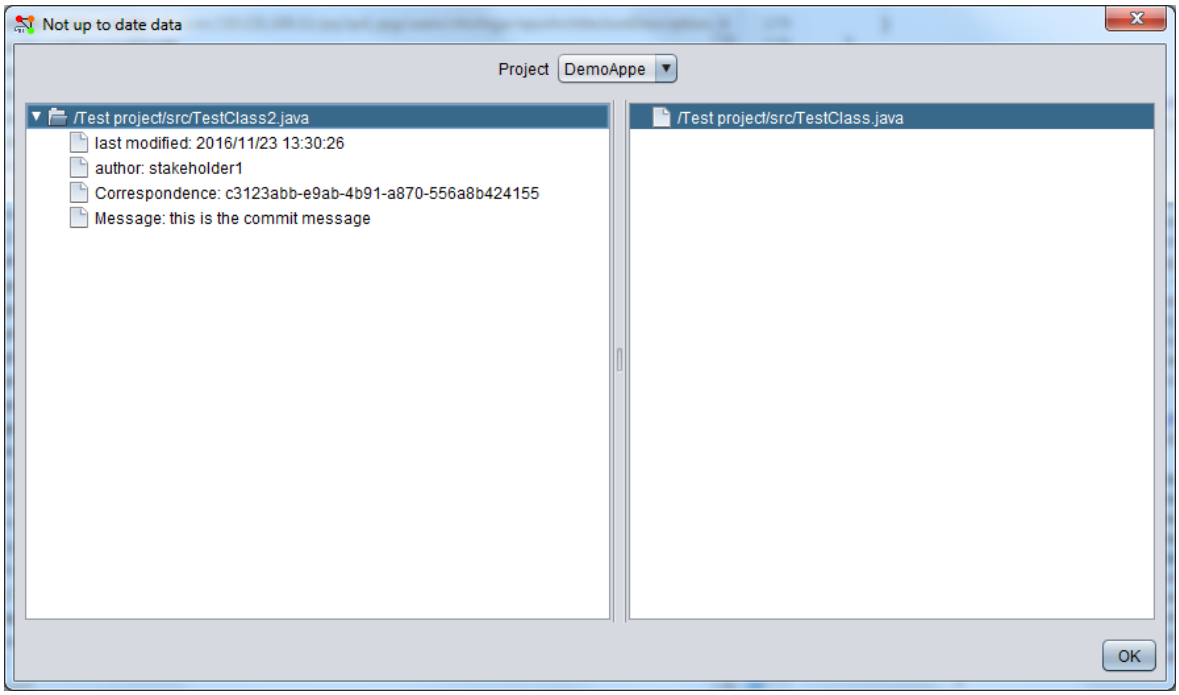


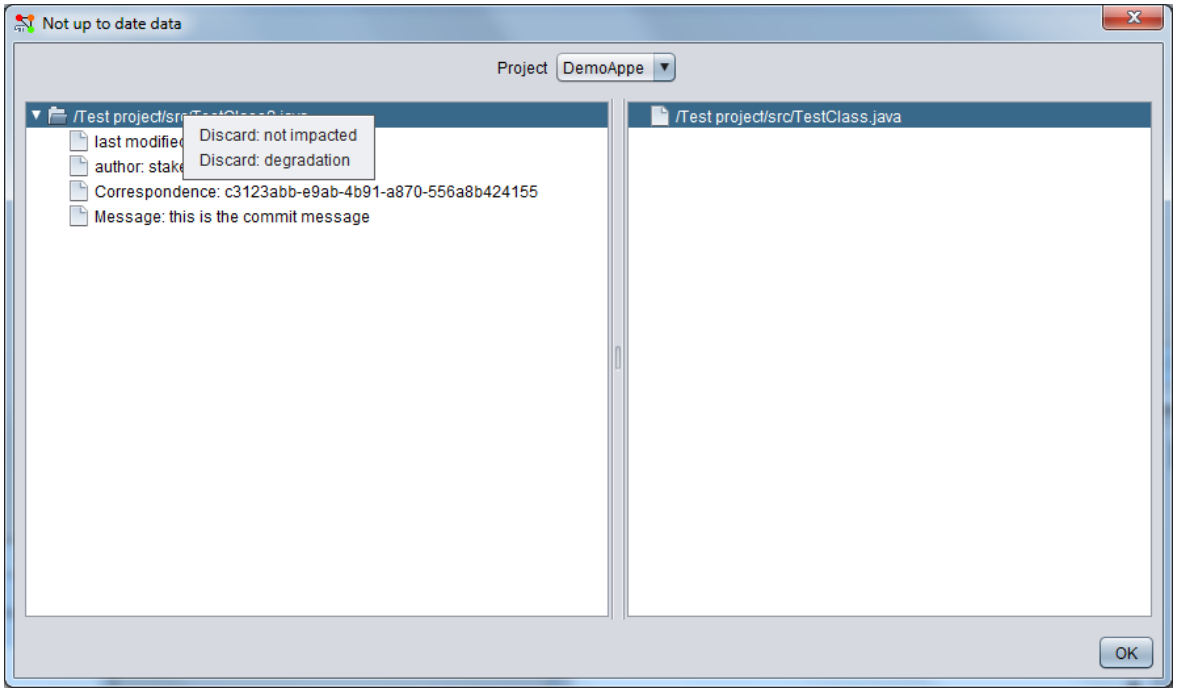
d. DialogCompatibility



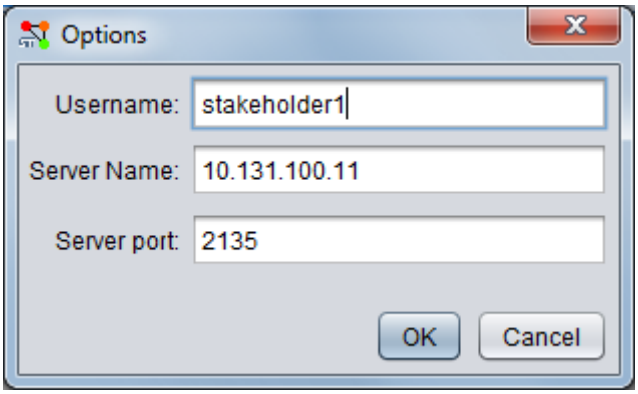


e. DialogNotUpdated

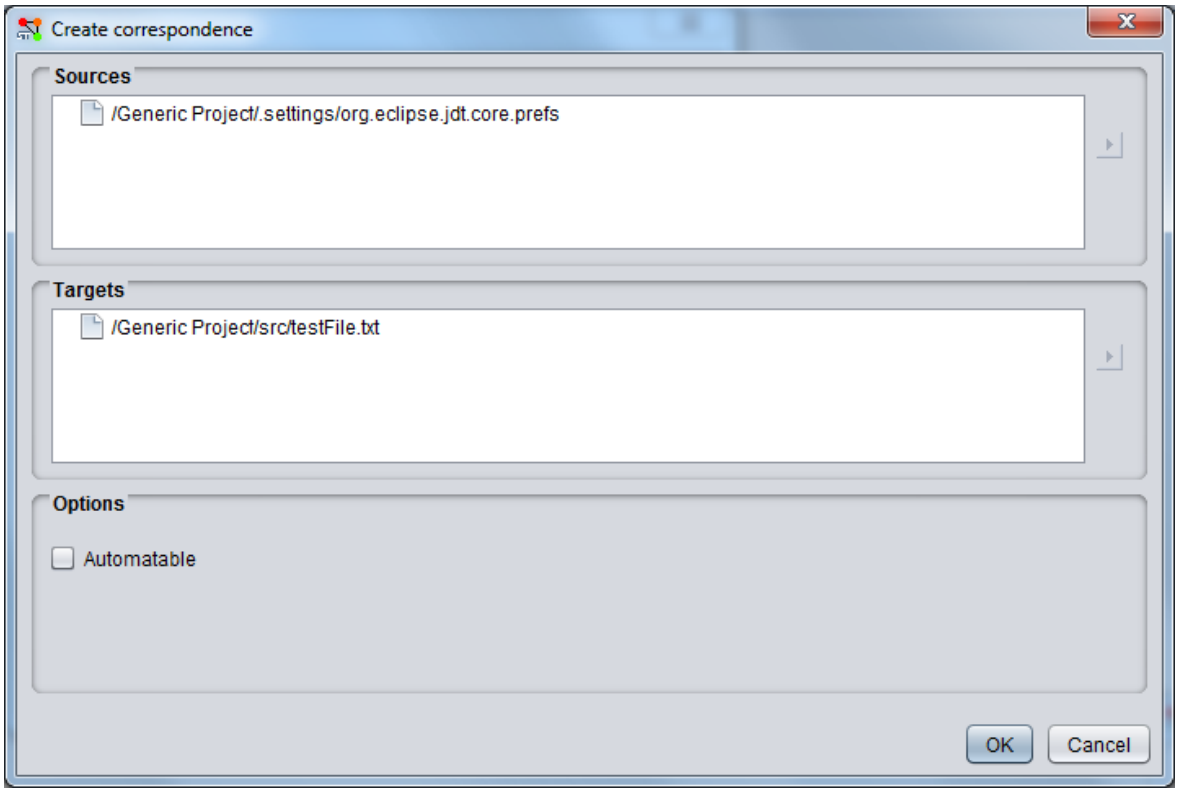




f. DialogOption

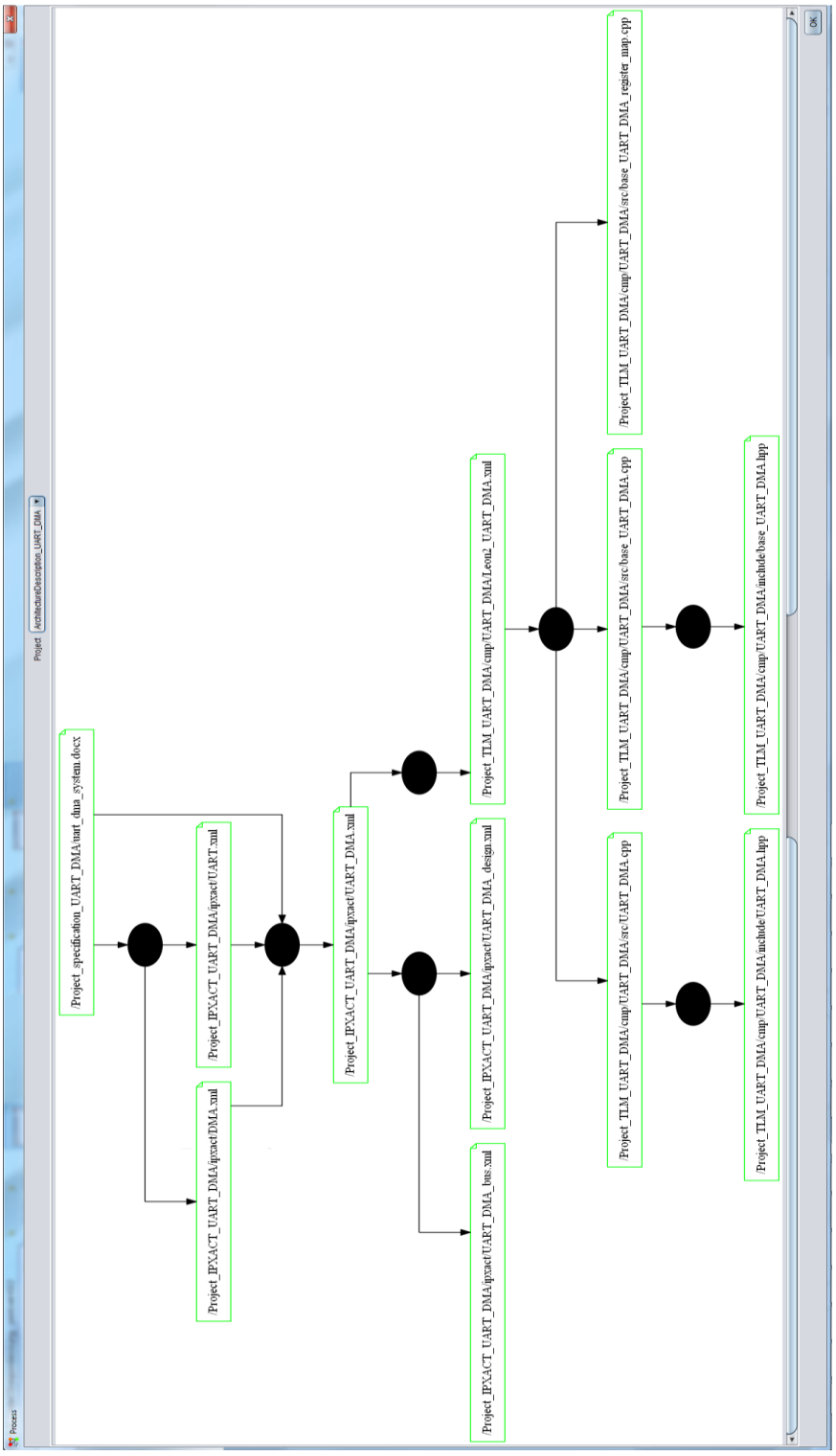


g. DialogCreateCorrespondence

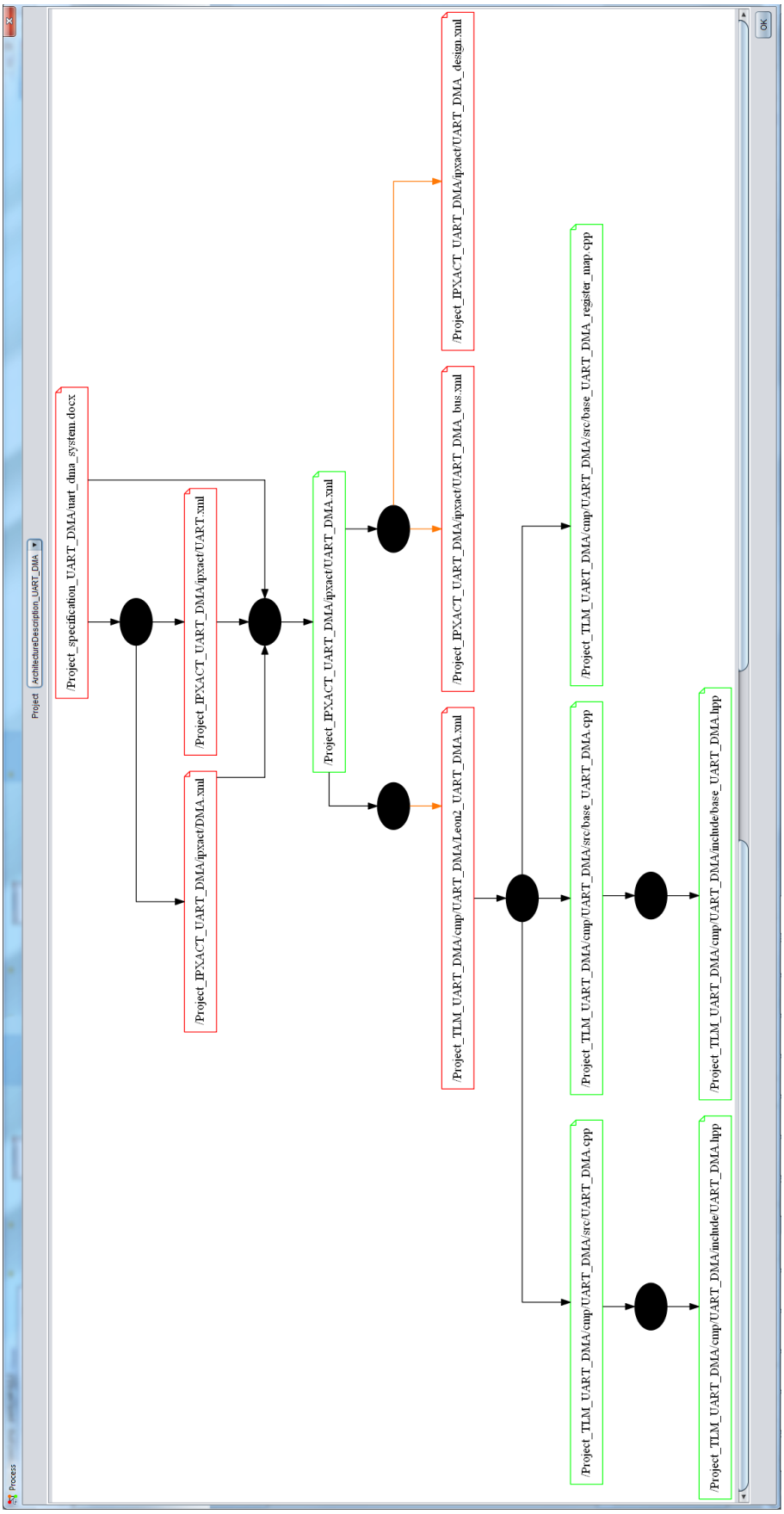


Annexe 6. Processus générés par APPE

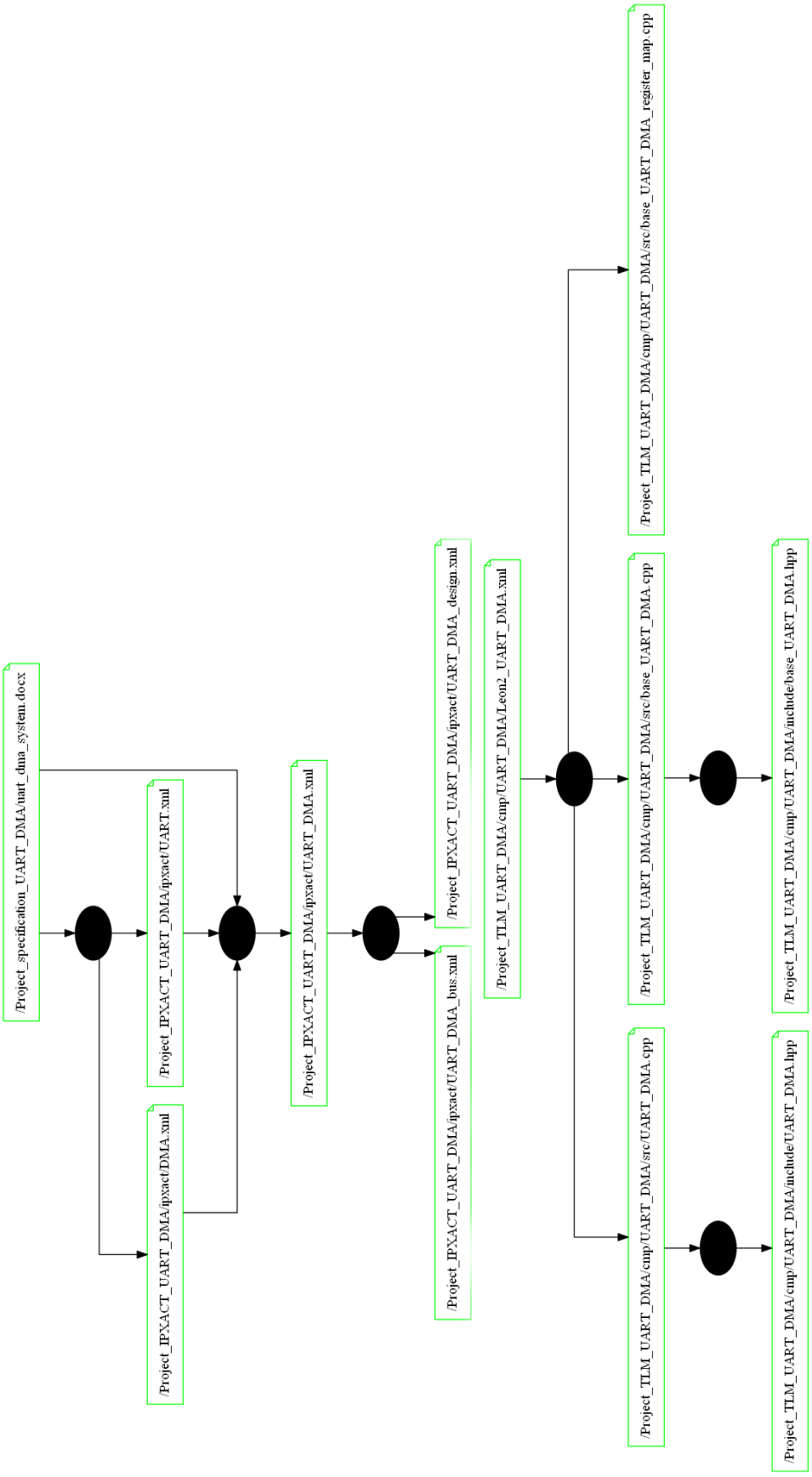
1. Section 6.2.2



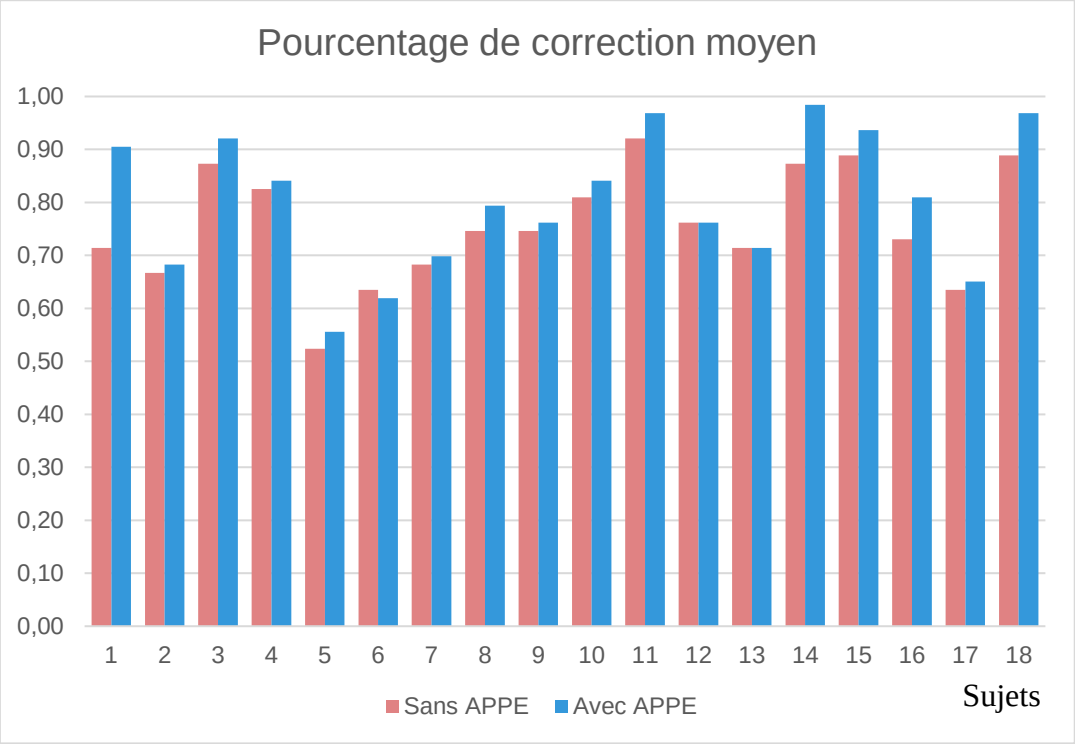
2. Section 6.2.3

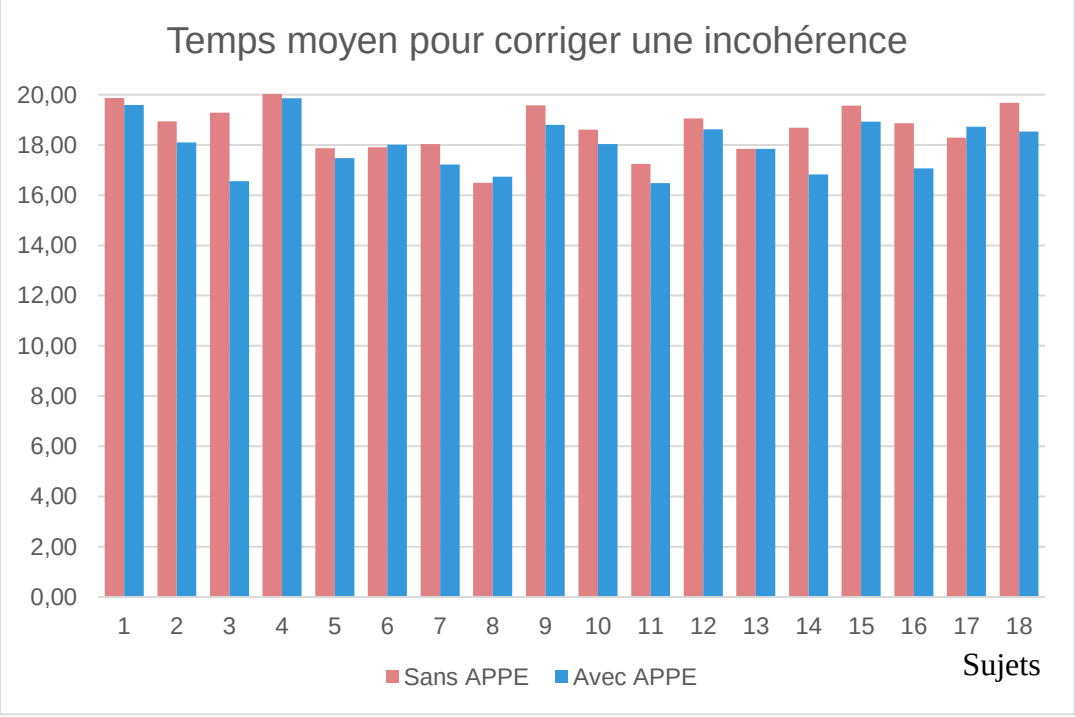


3. Section 6.2.3.3



Annexe 7. Résultats de l'étude





1. Utilisateur #1

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMA.xml	1	1	100		
	UART_DMA.xml	2	1	50		
	UART_DMA_bus.xml	1	0	0		
	UART_DMA_design.xml	4	2	50		
	Leon2_UART_DMA.xml	3	2	66.67		
	Result	12	7	58.33	120	17.14
2	UART_DMA.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMA.xml	3	3	100		
	base_UART_DMA_register_map.cpp	2	0	0		
	Result	11	9	81.82	120	13.33
3	UART_DMA.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	1	0	0		
	Result	4	3	75	73	24.33
4	UART_DMA.xml	1	1	100		
	DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	1	0	0		
	Result	4	3	75	68	22.67
5	uart_DMA_system.docx	4	2	50		
	UART_DMA.xml	2	2	100		
	Leon2_UART_DMA.xml	2	2	100		
	base_UART_DMA_register_map.cpp	2	0	0		
	Result	10	6	60	113	18.83
6	uart_DMA_system.docx	1	1	100		
	UART_DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	2	0	0		
	Result	5	3	60	70	23.33
7	uart_DMA_system.docx	1	1	100		
	UART_DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	1	0	0		
	Result	4	3	75	66	22.00
8	uart_DMA_system.docx	2	1	50		
	UART_DMA.xml	2	2	100		
	Leon2_UART_DMA.xml	2	2	100		
	DMA.xml	2	2	100		
	Result	8	7	87.5	120	17.14
9	uart_DMA_system.docx	2	1	50		
	UART_DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	DMA.xml	1	1	100		
	Result	5	4	80	80	20.00
	Total	63	45	71.43	92.22	19.87

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMA.xml	1	1	100		
	UART_DMA.xml	2	2	100		
	UART_DMA_bus.xml	1	1	100		
	UART_DMA_design.xml	4	3	75		
	Leon2_UART_DMA.xml	3	3	100		
		12	11	91.67	120	10.909091
2	UART_DMA.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMA.xml	3	3	100		
	base_UART_DMA_register_map.cpp	2	2	100		
		11	11	100	120	10.909091
3	UART_DMA.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	1	1	100		
		4	4	100	96	24
4	UART_DMA.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	1	0	0		
		4	3	75	83	27.666667
5	uart_DMA_system.docx	4	2	50		
	UART_DMA.xml	2	2	100		
	Leon2_UART_DMA.xml	2	2	100		
	base_UART_DMA_register_map.cpp	2	2	100		
		10	8	80	120	15
6	uart_DMA_system.docx	1	1	100		
	UART_DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	2	2	100		
		5	5	100	103	20.6
7	uart_DMA_system.docx	1	1	100		
	UART_DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	base_UART_DMA_register_map.cpp	1	1	100		
		4	4	100	100	25
8	uart_DMA_system.docx	2	0	0		
	UART_DMA.xml	2	2	100		
	Leon2_UART_DMA.xml	2	2	100		
	DMA.xml	2	2	100		
		8	6	75	120	20
9	uart_DMA_system.docx	2	2	100		
	UART_DMA.xml	1	1	100		
	Leon2_UART_DMA.xml	1	1	100		
	DMA.xml	1	1	100		
		5	5	100	111	22.2
	Total	63	57	90.48	108.111	19.587205

2. Utilisateur #2

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	0	0		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	6	50.00	120.00	20.00
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	1	33.333		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	9	81.818	120	13.33
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	66	22.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	62	20.67
5	uart_dmac_system.docx	4	0	0		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	6	60	109	18.17
6	uart_dmac_system.docx	1	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	4	80	82	20.50
7	uart_dmac_system.docx	1	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	2	50	38	19.00
8	uart_dmac_system.docx	2	0	0		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	6	75	109	18.17
9	uart_dmac_system.docx	2	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	3	60	56	18.67
	Total	63	42	66.67	84.67	18.94

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	0	0		
	DMAC.xml	1	0	0		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	9	75.00	120	13.33
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	10	90.91	120	12
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	77	19.25
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	74	18.5
5	uart_dmac_system.docx	4	1	25		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	4	40	86	21.5
6	uart_dmac_system.docx	1	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	3	60	58	19.33333333
7	uart_dmac_system.docx	1	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	59	19.66666667
8	uart_dmac_system.docx	2	0	0		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	3	37.5	58	19.33333333
9	uart_dmac_system.docx	2	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	3	60	60	20
	Total	63	43	68.25	79.11	18.10185185

3. Utilisateur #3

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	11	91.67	120.00	10.91
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	9	81.82	120	13.33
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	93	23.25
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	4	100	89	22.25
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	8	80	120	15.00
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	3	60	92	30.67
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	81	27.00
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	116	14.50
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	83	16.60
	Total	63	55	87.30	101.56	19.28

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100		
		12	12	100.00	110	9.166667
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	11	100	113	10.27273
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	81	20.25
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	81	20.25
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	6	60	112	18.66667
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	76	19
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	72	18
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	120	15
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	92	18.4
	Total	63	58	92.06	95.2222	16.55623

4. Utilisateur #4

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	0	0		
	DMAC.xml	1	0	0		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	9	75.00	120.00	13.33
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	0	0		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	8	72.73	120	15.00
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	87	29.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	85	28.33
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	10	100	118	11.80
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	103	20.60
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	84	21.00
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	0	0		
		8	6	75	120	20.00
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
		5	4	80	86	21.50
	Total	63	52	82.54	102.56	20.06

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	0	0		
	DMAC.xml	1	0	0		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100		
		12	10	83.33	120	12
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	0	0		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	8	72.73	120	15
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	80	26.66667
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	74	24.66667
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	10	100	120	12
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	108	21.6
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	95	23.75
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	0	0		
		8	6	75	120	20
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
		5	4	80	92	23
	Total	63	53	84.13	103.222	19.8537

5. Utilisateur #5

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	0	0		
	UART_DMAC_design.xml	4	0	0		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	5	41.67	86.00	17.20
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	0	0		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	6	54.54	106	17.67
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	2	50	37	18.50
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	2	50	36	18.00
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	0	0		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	4	40	76	19.00
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	2	40	33	16.50
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	2	50	36	18.00
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	0	0		
	DMAC.xml	2	2	100		
		8	6	75	102	17.00
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	DMAC.xml	1	1	100		
		5	4	80	76	19.00
	Total	63	33	52.38	65.33	17.87

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	0	0		
	UART_DMAC_design.xml	4	0	0		
	Leon2_UART_DMAC.xml	3	3	100		
		12	7	58.33	114	16.28571
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	3	27.27	61	20.33333
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	56	18.66667
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	56	18.66667
5	uart_dmac_system.docx	4	1	25		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	5	50	86	17.2
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	3	60	55	18.33333
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	48	16
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	4	50	67	16.75
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	60	15
	Total	63	35	55.56	67	17.47063

6. Utilisateur #6

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	9	75.00	109.00	12.11
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	9	81.82	103	11.44
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	57	19.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	52	17.33
5	uart_dmac_system.docx	4	1	25		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	3	30	54	18.00
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	58	14.50
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	68	17.00
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	0	0		
	Leon2_UART_DMAC.xml	2	0	0		
	DMAC.xml	2	0	0		
		8	1	12.5	35	35.00
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	67	16.75
	Total	63	40	63.49	67.00	17.90

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	9	75.00	108	12
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	9	81.82	106	11.77778
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	62	20.66667
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	57	19
5	uart_dmac_system.docx	4	1	25		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	3	30	50	16.66667
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	54	13.5
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	60	20
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	0	0		
	Leon2_UART_DMAC.xml	2	0	0		
	DMAC.xml	2	0	0		
		8	1	12.5	29	29
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	78	19.5
	Total	63	39	61.90	67.1111	18.01235

7. Utilisateur #7

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	0	0		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	6	50.00	118.00	19.67
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	9	81.82	120	13.33
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	53	17.67
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	47	15.67
5	uart_dmac_system.docx	4	1	25		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	7	70	108	15.43
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	3	60	54	18.00
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	2	50	47	23.50
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	120	15.00
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	DMAC.xml	1	0	0		
		5	2	40	48	24.00
	Total	63	43	68.25	79.44	18.03

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	0	0		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	6	50.00	120	20
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	9	81.82	116	12.88889
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	52	17.33333
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	61	15.25
5	uart_dmac_system.docx	4	1	25		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	7	70	110	15.71429
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	3	60	59	19.66667
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	2	50	48	24
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	105	13.125
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	DMAC.xml	1	0	0		
		5	2	40	34	17
	Total	63	44	69.84	78.3333	17.2198

8. Utilisateur #8

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	0	0		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	9	75.00	113.00	12.56
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	5	45.45	95	19.00
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	64	16.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	65	16.25
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	6	60	83	13.83
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	76	15.20
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	70	17.50
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	0	0		
		8	6	75	89	14.83
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	93	23.25
	Total	63	47	74.60	83.11	16.49

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	0	0		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	3	100		
		12	9	75.00	107	11.88889
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	5	45.45	93	18.6
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	81	20.25
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	63	15.75
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	6	60	90	15
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	80	16
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	81	20.25
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	106	13.25
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	98	19.6
	Total	63	50	79.37	88.7778	16.7321

9. Utilisateur #9

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	1	33.33		
		12	10	83.33	120.00	12.00
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	5	45.45	113	22.60
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	81	20.25
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	76	19.00
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	120	15.00
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	73	18.25
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	85	21.25
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	5	62.5	86	17.20
9	uart_dmac_system.docx	2	0	0		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	3	60	92	30.67
	Total	63	47	74.60	94.00	19.58

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	1	33.33		
		12	10	83.33	120	12
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	4	36.36	99	24.75
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	73	18.25
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	76	19
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	116	14.5
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	86	17.2
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	82	20.5
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	5	62.5	86	17.2
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	103	25.75
	Total	63	48	76.19	93.4444	18.79444

10. Utilisateur #10

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	0	0		
	DMAC.xml	1	0	0		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	8	66.67	120.00	15.00
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	0	0		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	8	72.73	120	15.00
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	74	24.67
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	62	20.67
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	10	100	116	11.60
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	120	24.00
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	65	16.25
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	0	0		
		8	6	75	120	20.00
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
		5	4	80	81	20.25
	Total	63	51	80.95	97.56	18.60

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100		
		12	12	100.00	120	10
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	4	36.36	89	22.25
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	75	18.75
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	79	19.75
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	10	100	118	11.8
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	111	22.2
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	70	17.5
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	5	62.5	115	23
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	85	17
	Total	63	53	84.13	95.7778	18.02778

11. Utilisateur #11

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	12	100.00	110.00	9.17
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	10	90.91	106	10.60
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	97	24.25
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	98	24.50
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	103	12.88
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	100	25.00
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	70	17.50
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	1	50		
		8	7	87.5	89	12.71
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	93	18.60
	Total	63	58	92.06	96.22	17.25

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100		
		12	12	100.00	107	8.916667
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	11	100	112	10.18182
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	73	18.25
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	102	25.5
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	9	90	109	12.11111
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	98	24.5
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	70	17.5
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	115	14.375
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	85	17
	Total	63	61	96.83	96.7778	16.48162

12. Utilisateur #12

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	0	0		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	9	75.00	107.00	11.89
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	9	81.82	110	12.22
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	80	20.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	65	16.25
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	5	50	106	21.20
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	116	29.00
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	69	17.25
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	4	50	109	27.25
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	82	16.4
	Total	63	48	76.19	93.78	19.05

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	1	33.33		
		12	8	66.67	115	14.375
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	10	90.91	100	10
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	81	20.25
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	82	20.5
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	5	50	99	19.8
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	97	24.25
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	69	17.25
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	2	100		
		8	5	62.5	102	20.4
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	83	20.75
	Total	63	48	76.19	92	18.61944

13. Utilisateur #13

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	0	0		
	DMAC.xml	1	0	0		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	8	66.67	106.00	13.25
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	0	0		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	6	54.55	104	17.33
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	65	21.67
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	62	20.67
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	87	10.88
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	75	15.00
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	69	17.25
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	4	50	102	25.5
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	76	19
	Total	63	45	71.43	82.89	17.84

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	1	50		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	10	83.33	106	10.6
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	1	33.33		
	Leon2_UART_DMAC.xml	3	1	33.33		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	6	54.55	89	14.83333
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	75	18.75
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	74	18.5
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	5	50	83	16.6
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	94	23.5
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	69	17.25
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	4	50	89	22.25
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	73	18.25
	Total	63	45	71.43	83.5556	17.83704

14. Utilisateur #14

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	0	0		
	DMAC.xml	1	0	0		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	9	75.00	120.00	13.33
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	2	66.67		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	10	90.91	120	12.00
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	78	26.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	0	0		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	3	75	73	24.33
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	10	100	104	10.40
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	107	21.40
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	92	23
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	1	50		
		8	7	87.5	106	15.14285714
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
		5	4	80	90	22.5
	Total	63	55	87.30	98.89	18.68

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	11	91.67	116	10.54545
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	11	100	118	10.72727
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	96	24
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	87	21.75
5	uart_dmac_system.docx	4	4	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	10	100	107	10.7
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	98	19.6
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	95	23.75
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	102	12.75
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	88	17.6
	Total	63	62	98.41	100.778	16.82475

15. Utilisateur #15

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	2	50		
	Leon2_UART_DMAC.xml	3	2	66.67		
		12	9	75.00	120.00	13.33
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	11	100	120	10.91
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	113	28.25
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	112	28.00
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	120	15.00
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	112	22.40
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	66	22
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	115	14.375
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	87	21.75
	Total	63	56	88.89	107.22	19.56

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	3	75		
	Leon2_UART_DMAC.xml	3	3	100		
		12	11	91.67	120	10.90909
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	11	100	120	10.90909
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	102	25.5
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	113	28.25
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	120	15
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	115	23
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	87	29
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	105	13.125
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	73	14.6
	Total	63	59	93.65	106.111	18.92146

16. Utilisateur #16

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	12	100.00	110.00	9.17
2	UART_DMAC.xml	3	2	66.67		
	UART.xml	3	2	66.67		
	Leon2_UART_DMAC.xml	3	2	66.67		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	6	54.555	116	19.33
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	63	21.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	62	20.67
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	6	60	99	16.50
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	78	19.50
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	70	17.50
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	5	62.5	89	17.80
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	0	0		
		5	3	60	85	28.33
	Total	63	46	73.02	85.78	18.87

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100		
		12	12	100.00	115	9.583333
2	UART_DMAC.xml	3	2	66.67		
	UART.xml	3	2	66.67		
	Leon2_UART_DMAC.xml	3	2	66.67		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	7	63.64	105	15
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	83	20.75
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	76	19
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	1	50		
	base_UART_DMAC_register_map.cpp	2	1	50		
		10	6	60	103	17.16667
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		5	4	80	76	19
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	70	17.5
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	5	62.5	102	20.4
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	76	15.2
	Total	63	51	80.95	89.5556	17.06667

17. Utilisateur #17

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	1	25		
	Leon2_UART_DMAC.xml	3	1	33.33		
		12	7	58.33	97.00	13.86
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		11	7	63.64	102	14.57
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	54	18.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	56	18.67
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	6	60	50	8.33
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	3	60	53	17.67
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	54	13.5
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	6	75	54	9
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	0	0		
	Leon2_UART_DMAC.xml	1	0	0		
	DMAC.xml	1	0	0		
		5	1	20	51	51
	Total	63	40	63.49	63.44	18.29

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	1	25		
	Leon2_UART_DMAC.xml	3	1	33.33		
		12	7	58.33	105	15
2	UART_DMAC.xml	3	1	33.33		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	8	72.73	105	13.125
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	69	23
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	65	21.66667
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		10	6	60	103	17.16667
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	0	0		
		5	3	60	68	22.66667
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	0	0		
		4	3	75	67	22.33333
8	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	2	1	50		
	Leon2_UART_DMAC.xml	2	1	50		
	DMAC.xml	2	1	50		
		8	4	50	87	21.75
9	uart_dmac_system.docx	2	1	50		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	4	80	47	11.75
	Total	63	41	65.08	79.5556	18.71759

18. Utilisateur #18

Without APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100.00		
		12	12	100.00	120.00	10.00
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	1	50		
		11	10	90.91	120	12.00
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	100	25.00
4	UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	105	26.25
5	uart_dmac_system.docx	4	0	0		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	6	60	120	20.00
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	120	24.00
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	90	22.5
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	0	0		
		8	6	75	114	19
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	92	18.4
	Total	63	56	88.89	109.00	19.68

With APPE						
#	Impacted files	Occurrences	Change done	%	Time (s)	Time rate
1	UART.xml	1	1	100		
	DMAC.xml	1	1	100		
	UART_DMAC.xml	2	2	100		
	UART_DMAC_bus.xml	1	1	100		
	UART_DMAC_design.xml	4	4	100		
	Leon2_UART_DMAC.xml	3	3	100		
		12	12	100.00	120	10
2	UART_DMAC.xml	3	3	100		
	UART.xml	3	3	100		
	Leon2_UART_DMAC.xml	3	3	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		11	11	100	120	10.90909
3	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	102	25.5
4	UART_DMAC.xml	1	1	100		
	UART.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	112	28
5	uart_dmac_system.docx	4	2	50		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		10	8	80	120	15
6	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	2	2	100		
		5	5	100	104	20.8
7	uart_dmac_system.docx	1	1	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	base_UART_DMAC_register_map.cpp	1	1	100		
		4	4	100	95	23.75
8	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	2	2	100		
	Leon2_UART_DMAC.xml	2	2	100		
	DMAC.xml	2	2	100		
		8	8	100	112	14
9	uart_dmac_system.docx	2	2	100		
	UART_DMAC.xml	1	1	100		
	Leon2_UART_DMAC.xml	1	1	100		
	DMAC.xml	1	1	100		
		5	5	100	94	18.8
	Total	63	61	96.83	108.778	18.52879

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Approche méthodologique pour le maintien de la cohérence des données de conception des systèmes sur puce

Mots clés : cohérence, système sur puce, traçabilité, description d'architecture, analyse d'impact, agilité

Résumé : Le développement de produits complexes demande la maintenance d'un grand nombre de documents interdépendants exprimés dans différents formats. Malheureusement, aujourd'hui, aucun outil et aucune méthodologie ne nous permettent pas de maintenir la cohérence et de propager systématiquement les changements entre ces documents.

D'après les observations faites dans l'entreprise STMicroelectronics, lorsqu'un document est modifié, les développeurs doivent propager manuellement la modification à l'ensemble des documents impactés. Pour diverses raisons, ces changements peuvent ne pas être correctement appliqués, voir même ne pas être appliqués du tout. Les documents divergent alors peu à peu, impactant dramatiquement le temps de développement pour réaligner tous les documents.

Nous proposons une méthodologie aidant les développeurs à maintenir systématiquement la cohérence entre les documents, basée sur le concept de description d'architecture introduit par l'ISO42010. Premièrement, un modèle est défini pour décrire formellement et complètement des correspondances (liens existants) entre des documents. Ce modèle est défini pour être indépendant des formats de documents, du cycle de développement et des méthodes de travail de l'entreprise. Deuxièmement, ces correspondances sont analysées afin d'aider les développeurs à maintenir la cohérence des documents en les informant lorsqu'un document est modifié.

Un prototype mettant en œuvre l'approche proposée a été développé afin d'évaluer la méthodologie. 18 sujets se sont portés volontaires afin d'évaluer l'approche. Ces sujets ont été soumis à deux tests (avec et sans notre méthodologie) impliquant la correction d'incohérences ajoutées dans un ensemble de documents. Ces tests nous ont permis de dégager deux variables : le nombre d'incohérences corrigées et le temps moyen pour corriger les incohérences. Selon notre étude, l'utilisation de notre approche permet de corriger 5,5% d'incohérences en plus en un temps 3,3% plus faible.

Title: Methodological approach for maintaining consistency of system on chip design data

Keywords: consistency, system on chip, traceability, architecture description, impact analysis, agility

Abstract: The development of highly complex products requires the maintenance of a huge set of inter-dependent documents, in various formats. Unfortunately, no tool or methodology is available today to systematically maintain consistency between all these documents.

Therefore, according to observations made in STMicroelectronics, when a document changes, stakeholders must manually propagate the changes to the impacted set of dependent documents. For various reasons, they may not well propagate the change, or even may not propagate it at all. Related documents thereby diverge more and more over time. It dramatically impacts productivity to realign documents and make the very wide-ranging corpus of documents consistent.

This paper proposes a methodology to help stakeholders to systematically maintain consistency between documents, based on the Architecture Description concept introduced by ISO42010. First, a model is defined to describe formally and completely correspondences between Architecture Description Elements of documents. This model is designed to be independent of documents formats, selected system development lifecycle and the working methods of the industry. Second, these correspondences are analyzed in case of document modification in order to help stakeholders maintaining global corpus consistency.

A prototype has been developed, which implements the proposed approach, to evaluate the methodology. 18 subjects volunteered to evaluate the approach. These subjects made two tests (with and without our methodology) involving the correction of inconsistencies added in a set of documents. These tests allowed us to identify two variables: the number of inconsistencies corrected and the average time to correct the inconsistencies. According to our study, the use of the approach helps to correct 5.5% more inconsistencies in a time 3.3% lower.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France