Prolégomènes à une critique du génie logiciel[1]

Partie III : Interfaces

par Ivan Maffezzini, Alice Premiana et Bernardo Ventimiglia

La complexité [de certains systèmes] *viole les frontières traditionnelles entre les disciplines et demande une approche interdisciplinaire.*
Mario Bunge *, A World of Systems*

Je n'ai rien à faire avec le problème de la liberté métaphysique. Savoir si l'homme est libre ne m'intéresse pas. Je ne puis éprouver que ma propre liberté.
Albert Camus*, le Mythe de Sisyphe*

Très peu d'utilisateurs sont vraiment conscients ou sont capables d'articuler leurs buts.
Alan Cooper*, About Face 2.0*

La majorité a toujours tort.
T. W. Adorno*, Minima Moralia*

Résumé : : Le présent article est le troisième d'une série de quatre où nous tâchons de montrer que le *génie logiciel* (GL) ne peut aspirer à devenir l'un des centres – robustes – des activités d'automatisation que si les branches hypertrophiées par les vicissitudes historiques et les conflits économiques sont élaguées. Dans le premier article, nous avons « arbitrairement » fixé les principes qui soutiennent notre démarche. Dans le deuxième, nous avons essayé de mettre en évidence un certain manque dans le domaine de la qualité et des mesures logicielles. Dans cet article, nous considérerons l'automatisation du point de vue des interfaces, en particulier du point de vue des connaissances dont on a besoin pour les spécifier. Après avoir proposé une classification, nous analyserons en détail les interfaces machine-machine en nous appuyant sur des exemples d'interfaces normalisées. Suivra une analyse des interfaces personne-machine, personne-personne et machine-nature. Trois principes très généraux sont introduits pour compléter ceux du premier article. Cet article ne veut rien démontrer mais seulement montrer qu'il existe peut-être d'autres voies que celles qui sont actuellement suivies par la majorité pour cerner le génie logiciel

1. Introduction

Dans tous les domaines techniques, les interfaces ont une importance fondamentale, car ce sont elles qui permettent de réaliser des machines complexes composant par composant : de diviser pour régner. Dans cet article, nous nous proposons de considérer l'automatisation du point de vue des interfaces, dans le but de voir si, à partir de cet angle, nous pouvons mieux saisir les difficultés du génie logiciel (GL) dans le processus d'automatisation. Cet angle devrait surtout nous permettre de mieux cerner la problématique des relations entre les différents types de spécifications, les connaissances requises pour les produire et le GL. Nous nous appuierons sur la définition d'interface présentée en [1]. Avec cet article, nous ne voulons rien démontrer mais seulement montrer qu'en prenant d'autres sentiers, moins battus, on peut voir certains aspects nouveaux du génie logiciel, ce qui peut aider à penser d'autres manières de le maîtriser.

Les quatre acceptions d'*interface* de [1] peuvent être réduites à deux, les deux dernières n'étant que l'application au verbe (*to interface*, en anglais) des deux premières acceptions qui s'appliquent au substantif (*Interface*). La première acception :

- *Une limite partagée à travers laquelle transite l'information*

[1] Nous remercions pour les commentaires et les critiques très pertinentes Guy Tremblay, Louis Martin et Philippe Gabrini professeurs à l'université du Québec à Montréal.

ne considère pas le support physique qui permet l'échange, tandis que la deuxième en tient compte :

- *Un composant matériel ou logiciel qui fait la jonction entre deux autres composants dans le but de faire transiter l'information entre l'un et l'autre.*

Les deux acceptions sont donc étroitement liées et nous passerons de l'une à l'autre en fonction du contexte.

Dans le domaine de la technique, la définition d'un concept en langue naturelle est rarement utile s'il n'est pas accompagné d'un mécanisme quelconque de classification. En ce sens, toute classification est un moyen qui facilite la maîtrise conceptuelle d'un domaine. Une classification, tout en constituant un point de vue arbitraire, doit être systématique et rigoureuse si on veut l'appliquer efficacement dans un domaine technique. Étant donné qu'il s'agit d'un point de vue, il est assujetti aux aléas de la progression technique et ne peut être rendu immuable que par un choix normatif, arbitraire lui aussi. Si, en raison de l'évolution technique ou de l'approfondissement conceptuel, le domaine change, la classification peut changer aussi. Par exemple, la découverte des virus a entraîné des débats dans la classification du vivant, comme la nouvelle vision de l' « animalité » a changé la taxinomie des mammifères supérieurs[2].

Dans le domaine du génie logiciel (GL), la classification des interfaces n'a jamais posé de réels problèmes. Ceci nous semble dû d'une part à la simplicité du concept et de l'autre à un manque d'approfondissement. Manque d'approfondissement qui est sans doute dû aussi au consensus entre les théoriciens et les praticiens du génie logiciel qui ont des problèmes bien plus graves à régler[3].

En ce qui concerne la définition d'interface, nous n'avons aucun problème à retenir celle d'IEEE proposée en [1]. Pour ce qui a trait à la classification, nous présenterons celle qui est implicitement présente dans plusieurs normes d'IEEE et en particulier en [2]. Nous soulèverons certaines réserves par rapport à cette classification et nous en proposerons une autre qui nous semble plus utile et plus efficace pour un travail critique des fondements du GL.

Avant de considérer la classification des interfaces, nous allons introduire deux définitions et une note tirées de [3] car il s'agit d'éléments qui sont à la base de notre démarche.

> **Machine** : *Un artefact humain composé de parties nécessaires à la réalisation des fonctions qui ont présidé à sa construction. Une partie d'une machine peut être une machine.*
>
> **Automatisation** : *Emploi de machines pour transformer des données reçues de l'extérieur et exécuter des actions autonomes ayant un impact sur le monde.*
>
> **NOTE** : *à cause de leur ambiguïté et des mauvaises compréhensions qu'ils ont engendrées dans la brève histoire de l'automatisation assistée par ordinateur, nous tâcherons d'éviter l'emploi des termes* système*,* symbole *et* problème.

En tant qu'artefact humain, une machine a une ou plusieurs limites (frontières) qui donnent à la machine son unité et qui lui permettent d'échanger des données avec tout ce qui lui est extérieur. En d'autres termes, les interfaces président à la naissance même de la machine. Quand on parle d'une machine du point de vue fonctionnel, que voulons-nous dire sinon que la machine *est ce qu'elle est* à cause des échanges de données qui lui permettent l'exécution des fonctions ? Si on considère une machine constituée de machines, constituées de machines, etc. chaque machine interne, indépendamment de son niveau, a des interfaces qui lui permettent des échanges de données avec les autres machines et qui la caractérisent fonctionnellement. Les caractérisations fonctionnelles des machines internes sont influencées par la manière de structurer la machine externe, des interfaces donc ou, pour employer un terme galvaudé en GL, par le « comment » de la

[2] Un problème classificatoire amusant, plus proche des interfaces car il concerne les douanes, c'est-à-dire les postes de contrôle pour les frontières (les interfaces) entre les États, souleva un long débat à la fin des années vingt du XX^{e} siècle aux douanes new-yorkaises. Fallait-il classifier la sculpture l'*Oiseau dans l'espace* de Constantin Brancusi comme objet d'art, objet manufacturé ou matière première ? Le débat n'avait bien sûr rien de théorique mais concernait la taxation.

[3] À ce propos, il est intéressant de voir que les livres canoniques en génie logiciel, d'auteurs comme Sommerville, Ghezzi, Presmann, Booch, ne traitent avec une certaine profondeur que des interfaces entre modules (classes et méthodes).

conception. Ce qui montre que la division entre le « quoi » et le « comment » n'a pas de signification si l'on ne la situe pas dans un contexte précis via des interfaces.

2. Classification

Une première division canonique des interfaces est celle entre interfaces externes et interfaces internes ; il s'agit de la division qui sépare tout ce qui concerne les exigences décrites dans la spécification des exigences du logiciel [2] ou du système des exigences de conception. Les interfaces externes sont donc du ressort de la première partie du cycle de vie du logiciel et les interfaces internes de la deuxième partie « plus technique ». Cette division, qui a l'air si naturelle, nous semble n'être d'aucune utilité dans la classification des interfaces, car toutes les interfaces sont externes par définition puisqu'elles définissent le comportement externe d'une machine. Que cette première division « aille de soi » est dû au fait que c'est la connotation contractuelle qui prime et qu'un contrat a besoin de la définition de la frontière de la machine à acheter. C'est aussi une nécessité technique de développement, car la définition des frontières équivaut à une définition fonctionnelle de ce que la machine doit faire. Mais cela ne suffit pas, à notre avis, pour lui donner assez d'importance dans une étude des interfaces en raison de son manque de spécificité.

Si nous considérons la table des matières proposée en [2], nous avons implicitement une classification des interfaces externes de la machine[4] qui doit être réalisée en :

1. *Interfaces utilisateurs*
2. *Interfaces matérielles*
3. *Interfaces logicielles*
4. *Interfaces de communication*

Il est difficile de saisir ce qui a déterminé ce type de classification. On pourrait se demander pourquoi la différence entre interface matérielle et logicielle a été jugée à ce point importante qu'elle a une place à côté de l'interface utilisateur. Ou encore : pourquoi parler d'interfaces de communication à ce niveau ? Si on considère le mot communication d'une manière très générale, toutes les interfaces sont des interfaces de communication. Si on considère la communication du point de vue des réseaux de transmission de données, comme il nous semble qu'il faille le faire, nous estimons que son importance est exagérée, au moins au stade actuel de la technologie.

Nous proposons donc une classification avec une hiérarchie à deux niveaux, où le discriminateur du premier niveau est le langage et celui du second l'artificialité de l'artefact. La figure suivante montre le type d'entité avec laquelle un élément peut s'interfacer.

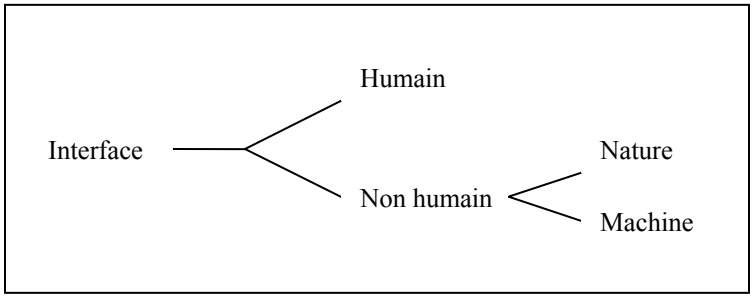

Figure 1 : Classification des interfaces

Cette classification, arbitraire comme celle des normes IEEE, a un défaut que celle des normes IEEE n'a pas : elle est inusitée en GL. Il faut donc expliquer les motivations de notre choix, pas tellement pour convaincre les lecteurs de sa validité, mais pour montrer qu'elle n'est pas fruit du hasard. Ce que nous espérons, c'est que la suite de l'article montrera que ce choix facilite une analyse critique du GL.

Le premier discriminateur permet de départager les humains du reste. Ce choix a été déterminé par l'importance que la langue a, à notre avis, dans la différentiation des manières de passer l'information et

[4] « Système » dans la terminologie IEEE.

donc dans le processus d'automatisation. Avant de considérer le discriminateur « langue », nous avions considéré l'« animalité » (interface avec les animaux et interface avec les non animaux), mais nous y avons renoncé parce que, dans l'état actuel de la technologie, nous estimons que la cohésion entre les caractéristiques des animaux humains et des autres animaux n'est pas assez forte. Il est par contre clair que la division à laquelle nous avons renoncé a un intérêt politique et social sans doute plus grand que celui de la division que nous avons adoptée. La branche « humain » n'a pas été ultérieurement divisée, car les caractéristiques propres aux différents types d'humains sont inessentielles à ce niveau d'analyse où nous en sommes.

La branche « non humaine » a été divisée en Nature et Machine. Nous avons regroupé en un seul taxon, que nous avons appelé *Nature*, les animaux, les végétaux et tout le reste de la réalité matérielle, c'est-à-dire tout ce qui existe indépendamment de l'homme. La nature s'oppose aux machines, les artefacts des humains. De *Nature*, nous avons toutefois décidé d'enlever les animaux et les végétaux, d'une part parce que leur importance actuelle est relative et de l'autre parce que nous n'avons pas la prétention d'avoir la capacité de les aborder.

Les trois taxons que nous avons conservés (Humains, Machine et Nature) permettent donc les types d'interfaces cités dans la table suivante.

Nom	**Remarque**
Personne-Machine	
Personne-Nature	D'aucune intérêt direct pour l'automatisation.
Personne-Personne	Importante pour les activités de travail en groupe.
Machine- Nature	La nature sans animaux et sans végétaux. La nature non vivante. Par exemple une interface avec l'air pour mesurer la température ambiante.
Machine-Machine	
Nature-Nature	Division qui intéresse la science mais pas l'automatisation.

Table 1 : Types d'interfaces

Nous nous retrouvons donc avec quatre types d'interfaces dont trois ont une machine.

Notre classification permet de subsumer tous les éléments de la classification d'IEEE :

IEEE		**Notre classification**	**Commentaire**
Externes			
	Utilisateur	Personne-machine	Pas de terme système.
	Matériel	Machine-machine, machine-nature	La nature dans le sens simplifié que nous avons retenu.
	Logiciel	Machine-machine	Le logiciel est une machine.
	Communication	Machine-machine	
		Personne-personne	
Internes		Machine-machine	Matériel ou logiciel, peu importe.

Table 2 : Mise en correspondance des classifications

Même si la mise en correspondance de la catégorie machine-machine avec quatre catégories d'IEEE peut faire penser à un plus faible degré d'analyse, nous croyons que c'est exactement le contraire : en poussant l'analyse, l'on s'aperçoit que les différences sont inessentielles. Par contre, le fait que l'interface matérielle ait, dans notre catégorisation, deux taxons souligne la différence de fond entre le matériel construit par les humains et le matériel naturel : au matériel construit par les humains (les machines, par exemple), on peut imposer des lois ; dans le cas du matériel naturel, c'est généralement lui qui nous impose ses lois.

3. Éléments constitutifs

Puisque l'interface permet un passage d'information, il faut qu'il existe un support physique qui transporte l'information. Nous l'appellerons le substrat de l'interface. Dans le cas de deux machines matérielles, le substrat peut être un connecteur avec ses caractéristiques mécaniques et électriques ou l'espace entourant la machine pour des liens infrarouges ou un être humain pour le transport d'un CD. Dans le cas de deux machines logicielles, de deux procédures, par exemple, il peut s'agir de la pile où le CPU, piloté par une procédure, dépose des données et que le CPU, piloté par l'autre, retire. Souligner que même les interfaces entre machines logicielles ont besoin d'un support matériel pour échanger de l'information n'est pas un excès de pédantisme mais une nécessité permettant de séparer ce qui se passe à l'exécution de ce qui se passe lors de la conception.

Avec les technologies actuelles, le substrat a souvent une composante mécanique et une composante électrique et électronique. À cause de la « physicité » qui lui est propre, la conception du substrat ne relève pas du domaine du génie logiciel et ceci même dans le cas des interfaces entre machines logicielles qui relèvent de l'ingénierie des ordinateurs et de l'électronique. Pour souligner encore davantage les caractéristiques du substrat dans les interfaces logicielles, on pourrait dire qu'une pile et le CPU sont l'équivalent d'un câble et de son connecteur.

Par-dessus le substrat il y a la partie fonctionnelle de l'interface qui, pour faciliter l'entretien, est souvent divisée en couches, comme il est montré dans la figure ci-dessous.

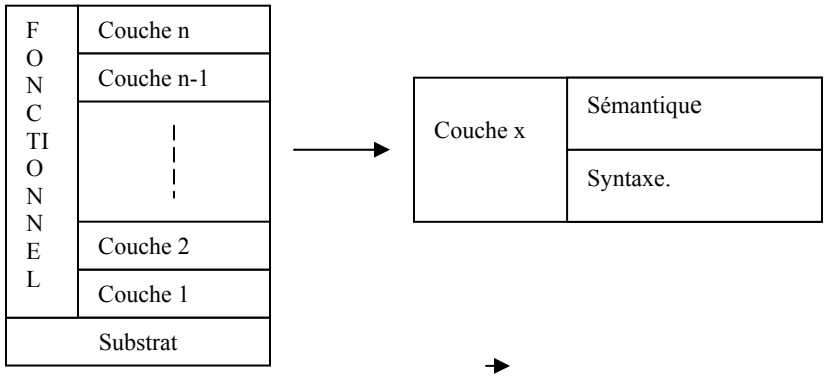

Figure 2 : Composition des interfaces

Chaque couche est à son tour divisée en deux parties : partie syntaxique (la forme de l'information) et partie sémantique (la signification de l'information).

Le modèle OSI est un bon exemple de définition d'interfaces ayant un substrat et une partie fonctionnelle divisées en couche, l'interface de chaque couche étant définie dans l'entête de l'unité de donnée de protocole[5].

Avant de conclure cette section, nous introduisons deux principes que nous appellerons les *principes fondamentaux des interfaces*. Principes qui, dans leur extrême simplicité, sont souvent oubliés et qui seront les guides de notre analyse :

> *Pour qu'il y ait échange d'information, il faut un substrat* (P06)[6].

[5] Dans le modèle OSI, les interfaces sont plus complexes car il y a aussi la définition des services (les interfaces entre les couches du même ordinateur). Dans notre classification, les interfaces entre les couches (les machines logicielles) de la même machine sont traitées comme les interfaces entre les couches des machines éloignées.

[6] Nous continuons la numérotation commencée avec le premier article des *Prolégomènes*.

Pour spécifier une interface il faut des connaissances dans le domaine dans lequel la machine opère (P07).

Il est important de noter que le principe P07 s'applique aux interfaces indépendamment du fait qu'elles soient externes ou internes. Ceci découle de notre approche où *externe* et *interne,* tels que considérés par IEEE et la majorité des auteurs, n'ont aucune importance

4. Les interfaces machine-machine

Nous ne considérerons que les interfaces entre deux machines, les interfaces entre plusieurs machines pouvant être ramenées, plus ou moins aisément, à plusieurs interfaces entre deux machines. Nous appellerons M1 la machine que nous devons réaliser et M2 celle à laquelle M1 doit être connectée. De la définition de machine découle que ses interfaces sont rigides et déterministes et peuvent être spécifiées de façon non ambiguë. Une spécification ambiguë sera donc le résultat d'erreurs humaines et non d'une impossibilité théorique.

Nous raffinons la classification des interfaces machine-machine en employant comme discriminateur les éléments ayant un impact sur le projet mis en place pour réaliser la machine M1. Comme premier discriminateur, nous prenons l'existence d'une spécification des interfaces et, comme deuxième, l'existence de la machine M2 au moment du début de la conception de M1. Du point de vue de la conception de M1, lorsqu'une spécification de l'interface existe, que M2 existe ou non est sans importance, si ce n'est, éventuellement, pour les essais. Dans le cas où la spécification n'existe pas, l'existence de la machine M2 devient importante.

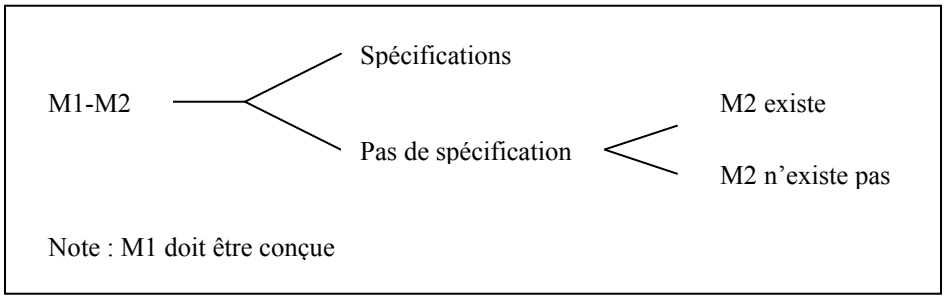

Figure 3 : Classification des interfaces machine-machine

4.1. La spécification existe.

Il s'agit du cas le plus simple parmi les trois possibles. La figure suivante présente les interactions entre les machines, les relations entre les spécifications et les relations entre les spécifications et les machines.

La figure est divisée en deux parties, la partie supérieure présente les spécifications de l'interface avec les flux d'informations lors de la conception ; la partie inférieure présente l'échange entre les machines lors du fonctionnement. Pour les deux machines, trois spécifications sont théoriquement nécessaires :

1. La spécification de l'interface (*Spéc. Interface*). Il s'agit de la description de l'interface selon la première acception de la définition d'IEEE, la spécification dont nous avons supposé l'existence avant la conception de M1. La norme RS 232-C, par exemple.
2. La spécification de l'interface de la machine M1 (*Spéc. Int. M1*). Il s'agit de la description du composant M1 considéré comme une interface selon la deuxième acception de la définition d'IEEE. Cette spécification, dans le cas le plus simple, ne fait que référer à *Spéc. Interface*. Le concepteur de M1 peut généraliser certains éléments, pourvu qu'il existe un sous-ensemble qui respecte les spécifications. Ce qui est certain, c'est qu'elle doit surtout concrétiser certains éléments pour rendre la réalisation possible. Si on considère l'exemple de *Spéc. Interface* la norme RS 232-C, celle-ci spécifie, par exemple, le nombre de broches, leur distance, la forme du connecteur… mais ne spécifie pas où le connecteur est fixé, avec quelles vis… ce que *Spéc. Int. M1* doit spécifier pour que l'on puisse construire la machine.

3. L’équivalent de *Spéc. Int. M1* mais pour M2.

Nul besoin de faire des hypothèses sur la langue de *Spéc. Interface*, pourvu qu’elle permette une description non ambiguë. Très souvent, surtout quand il s’agit de normes internationales, la description est un mélange d’une langue naturelle et d’une langue formelle, c’est-à-dire d’une langue avec une sémantique limitée et définie, proche du domaine où l’interface s’inscrit.

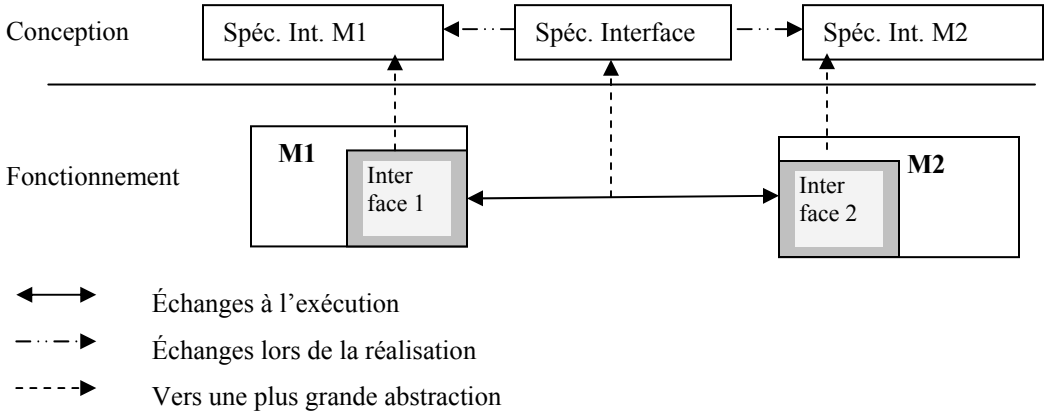

Figure 4 : Mise en correspondance d'exécution et conception I

Lors de la conception de la machine M1, le concepteur crée, en partant de *Spéc. Interface,* la spécification *Spéc. Int. M1*. La langue de *Spéc. Int. M1* n'est pas nécessairement la même que celle de *Spéc. Interface.* Dans le fait de changer de langue, il y a un côté négatif (dans la traduction des erreurs peuvent se glisser, erreurs qui s'ajoutent aux erreurs de programmation) et un autre positif (elle force le concepteur-traducteur à mieux s'assurer de la compréhension de la spécification originale[7]). Peu importe que le concepteur traduise ou non, ce qui est certain, c'est qu'il doit connaître la syntaxe et la sémantique de la langue employée pour décrire l'interface. La syntaxe peut être plus ou moins complexe, mais elle ne constitue jamais un « vrai » problème : elle demande un apprentissage plus ou moins long mais elle est abordable dans des temps assez brefs. La syntaxe est souvent assez générale et indépendante du domaine.

Ce n'est pas le cas pour la sémantique, c'est-à-dire pour la signification des informations qui transitent. Pour comprendre l'interface, il faut donc comprendre les concepts du domaine. Mais quel domaine ? La question n'est pas anodine comme on pourrait le penser de prime abord et elle se résume à la suivante : dans quel domaine l'interface est-elle insérée ? Voilà une question qui n'a pas de réponse simple. Rien de plus erroné que la réponse que les ingénieurs du logiciel donnent lorsque c'est une machine logicielle qui est en jeu : l'ingénieur du logiciel.

À l'aide des disciplines définies en SWEBOK et que nous avons synthétisés en [3] et reproduites ci-dessous avec les changements[8] dus aux modifications de la dernière version [4] et de trois exemples qui couvrent un spectre assez vaste, nous allons essayer de trouver une méthode pour fixer les domaines qu'il faut connaître.

[7] Il y a aussi l'impact non négligeable sur le codage. La présence de deux spécifications, même si, en général, c'est celle de sa propre interface qui prime, permet aux codeurs, dans les cas « difficiles » de considérer deux points de vue.

[8] *Sciences cognitives et facteurs humains* ont été renommés *Ergonomie logicielle* (une contradiction dans les termes, selon notre approche) et une nouvelle discipline a été introduite : *gestion de la qualité*.

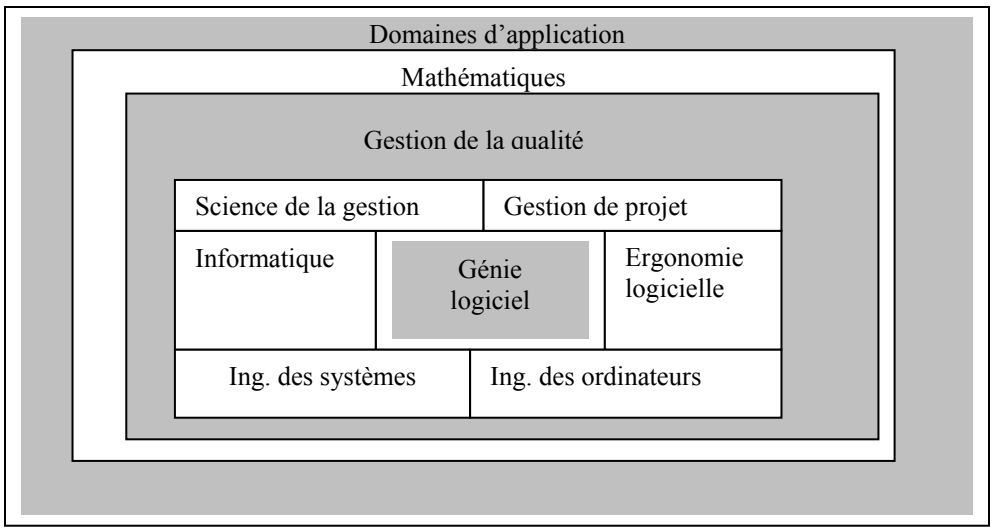

Figure 5 : Génie logiciel et domaines connexes

Les trois exemples choisis sont :

1. RS 323C : interface entre deux machines matérielles.
2. API de Windows : interface entre deux machines logicielles dont une fait partie du système d'exploitation.
3. ACSI de la norme IEC 61850 : interface entre deux machines logicielles faisant partie du domaine d'application.

À cause de ce que nous avons affirmé à la section 3, nous ne considérons pas le substrat.

RS 232-C. La partie fonctionnelle est constituée d'une seule couche : la couche qui décrit la signification des signaux. Considérons, pour simplifier, un sous-ensemble de l'interface et en particulier le signal CTS (*Clear to Send*). CTS peut avoir deux états stables : *On* et *Off* qui correspondent à deux valeurs de tension fixées par le substrat. L'état *On* indique que le modem peut transmettre les données (porteuse en ligne) et l'état *Off* indique qu'il ne le peut pas. Le domaine fonctionnel dans ce cas-ci est celui de la communication analogique. Pour comprendre la spécification il faut connaître les caractéristiques fonctionnelles d'un modem, ce qui permet, par exemple, de comprendre la différence entre modem prêt (*data set ready* à *On*) et modem qui peut transmettre (CTS à *On*). Celui qui conçoit la machine M1 dotée d'une interface RS 232-C (nécessairement matérielle) doit connaître le domaine des communications qui est un domaine transversal par rapport au domaine dans lequel et pour lequel les machines échangent des informations (le domaine bancaire, par exemple). Selon la terminologie de la figure 4, il s'agit du domaine de l'ingénierie des systèmes et de l'ingénierie des ordinateurs. Cela semble bien normal, vu que RS 232-C est un simple moyen de transport physique contrôlé de l'information.

Supposons que M1 contienne une machine logicielle M1.1 qui gère la transmission des données en fonctions du CTS. On est donc en présence d'une nouvelle interface : celle qui existe entre la machine logicielle M1.1 et le matériel de la carte qui contient l'interface RS 232-C. Supposons aussi que la spécification de la nouvelle interface contienne les deux phrases suivantes, très réalistes :

`1) Le bit 1 du registre d'adresse B0 contient l'état du CTS.`

`2) Suite à une écriture, le contenu du registre d'adresse A0 est sérialisé sans aucune vérification des signaux RS-232C.`

Il est clair que le responsable de la conception de M1.1 ne peut rien faire s'il ne connaît pas le fonctionnement du modem. Il devra donc aller chercher l'information dans la spécification officielle de RS 232-C d'où il apprendra qu'avant de transmettre (d'écrire la donnée en A0) il doit lire le CTS (bit 1 de

l'adresse B0)[9]. Le concepteur de M1.1 à donc besoin de deux spécifications de deux interfaces pour pouvoir concevoir/coder sa machine, mais aucune des spécifications n'a été écrite par lui.

Imaginons que la deuxième phrase de la spécification citée ci-dessus ait été la suivante :

`2) Suite à une écriture le contenu du registre d'adresse A0 est sérialisé si CTS est à *On*.`

Cette description est très différente, car c'est le matériel qui gère le CTS. Est-ce que les connaissances que l'on doit avoir sont différentes par rapport à la première version ? Non. Le fait que l'interface ait changé pour donner plus de fonctions au matériel ne change pas les connaissances requises pour spécifier l'interface et pour la faire fonctionner.

API de Windows. Nous allons prendre comme exemple *GetMessage,* la fonction qui récupère un message de la queue des messages du fil d'exécution qui fait l'appel. La fonction a comme paramètre un pointeur vers la structure qui va recevoir le message, un identificateur de la fenêtre et deux autres paramètres sans importance pour notre propos. *GetMessage* retourne un booléen qui indique si le message a été reçu ou si l'utilisateur a quitté l'application[10]. La fonction est synchrone. L'interface est clairement spécifiée et pratiquement sans ambiguïté, surtout si on suit tous les hyperliens qui ajoutent des informations théoriquement non essentielles mais importantes pour la compréhension. Quelles sont les connaissances dont a besoin le concepteur d'un logiciel qui veut s'interfacer à *getMessage* ? Clairement de celles du domaine des systèmes d'exploitation (quelle est la différence entre une procédure synchrone et asynchrones ? Qu'est-ce qu'un fil d'exécution ?) et de l'environnement *Windows* (quelles opérations faut-il faire quand un utilisateur quitte l'application ?). Selon la définition de SWEBOK, il faudrait parler du domaine de l'informatique. Puisque la différence entre génie logiciel et informatique est assez floue, à cette étape-ci de notre réflexion, on pourrait dire qu'elle demande des connaissances dans le deux domaines.

IEC 61850. IEC 81850 est une norme internationale pour l'interopérabilité des dispositifs électroniques intelligents. Une partie d'IEC 61850 spécifie les interfaces pour que des machines exécutant des fonctions dans un poste de transport ou de distribution de l'énergie puissent interopérer. Les machines matérielles dialoguent via un réseau local, avec les douze protocoles normalisés présentés dans la table 3. Les douze protocoles ne sont que des « supports » pour permettre aux processus d'application d'échanger des informations dont la sémantique est connue et commune.

Couche OSI	**Spécification du protocole**
Application	ISO 9506-2:2003 ISO/IEC 8560 1996
Présentation	ISO/IEC 8823-1 1994 et ISO/IEC 8825
Session	ISO/IEC 8327-1 1997.
Transport	RFC 1006, RFC 792 et RFC 793
Réseau	RFC 791 et RFC 894
Liaison de données	RFC 894 ISO 8802.3 : 2001

Table 3 : Pile des protocoles pour IEC 61850

Selon notre définition d'interfaces, nous devons considérer les douze protocoles comme des spécifications d'interfaces entre les machines logicielles qui, dans chaque machine matérielle, gèrent lesdits protocoles. Encore une fois, pour chaque interface, nous devons nous demander qui spécifie les interfaces. Encore une

[9] Si le projet est bien mené, il est probable qu'il existe un guide d'utilisation de la carte dans laquelle on aura une phrase du genre : « Avant d'écrire la donnée à transmettre dans le registre d'adresse A0, le logiciel doit vérifier que le bit 1 du registre de l'adresse B0 soit à 1 (CTS On) ». Ceci est un exemple des simplifications qui peuvent être introduites quand on écrit les guides d'utilisation très tôt dans le projet.

[10] Pour le détail voir http://msdn.microsoft.com/library/psdk/winui/messques_10kl.htm.

fois, il est clair qu'il s'agit d'un expert du domaine de l'ingénierie des systèmes, sous-domaine des protocoles. En raison de l'importance que les protocoles ont toujours eue dans le génie logiciel, nous avons résumé, dans l'annexe A, les activités nécessaires pour définir, mettre en œuvre et réaliser un protocole.

On peut considérer que ces douze protocoles créent une machine virtuelle pour les échanges entre processus d'application (couche 8 si on veut étendre la terminologie OSI). Donc, si la spécification des protocoles[11], comme il est montré dans l'annexe A, ne relève pas du génie logiciel, que se passe-t-il avec la couche 8 qui n'est pas une couche de communication au sens classique du terme ?

Considérons trois attributs d'une classe d'interface simple comme XCBR (disjoncteur) :

- *Loc* : indique si le disjoncteur opère sans communication avec la machine responsable de l'automatisation[12].
- *Pos* : position du disjoncteur (ouvert, fermé, état intermédiaire, état mauvais).
- *CBOpCap* : capacités d'opération (aucune, ouvre, ouvre-ferme, ouvre-ferme-ouvre, ferme-ouvre-ferme-ouvre).

Non seulement les disjoncteurs, mais chaque attribut est lui-même un concept du domaine électrique et le fait que ce soit ces attributs-là et pas d'autres qui caractérisent un disjoncteur ne dépend absolument pas du logiciel. On pourrait nous rétorquer que XCBR est une classe, et une classe est loin du monde des disjoncteurs. Bien sûr, mais le concept de classe, avant d'être un concept informatique, était un concept qui a accompagné le raisonnement humain depuis des millénaires. Mais, même s'il s'agissait d'un nouveau concept, il est très facile à assimiler par un non informaticien quand on oublie les détails de mise en œuvre. La classe telle qu'employée en IEC 61850 n'est qu'une « image » des objets du domaine.

Un ingénieur du logiciel qui connaît la norme pourrait critiquer notre position en soulignant que *Loc* est à son tour une classe dont les attributs ne sont plus des objets du domaine mais des booléens, des entiers, des chaînes de caractères…Oui, mais les entiers, les booléens avant d'être le pain quotidien des ingénieurs du logiciel étaient celui des ingénieurs électriques, civils, mécaniques.[13]

4.2. La spécification n'existe pas

C'est un cas qui, théoriquement, ne devrait pas exister (qu'avant de réaliser une machine, il faille spécifier ses interfaces, c'est le minimum requis pour une ingénierie digne de ce nom) mais qui est en fait assez répandu. Ce cas est surtout fort utile pour travailler sur un cas extrême et vérifier si nos considérations tiennent.

4.2.1. La machine M2 existe déjà

Nous faisons une hypothèse ultérieure : *la définition de l'interface de M2 ne change pas pendant la spécification, la conception et la construction de M1*. Cette hypothèse ne limite pas notre démarche, car le cas de l'interface de M2 qui change est un cas particulier de M1 et M2 conçues en même temps, cas qui est décrit dans la section suivante.
Nous reprenons la figure 3 en l'adaptant à la nouvelle situation. La partie fonctionnement ne change pas et, dans la partie conception, la flèche qui relie *Spéc. Interface* à *Spéc. Int. M2* change de sens (maintenant elle va vers *Spéc. Interface*) et une flèche reliant *Spéc. Int. M à Spéc. Int. M1* est ajoutée.

[11] Il est par contre clair que la conception des logiciels qui mettent en œuvre les protocoles ne relève pas du domaine des protocoles.
[12] Nous adaptons les définitions pour les rendre facilement compréhensibles pour ceux qui ne connaissent pas le domaine de l'automatisation des postes et des centrales.
[13] On pourrait, avec bien plus de raisons, retourner complètement le raisonnement et dire que les ingénieurs du logiciel sont des ingénieurs parce que, entre autres, ils s'approprient des concepts comme celui d'entier et de booléen que les génies traditionnels avaient empruntés aux mathématiques.

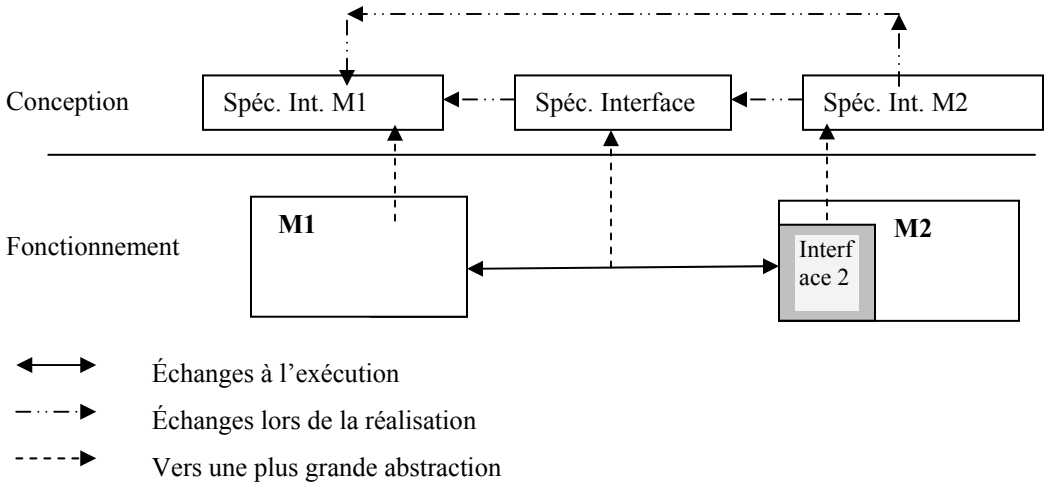

Figure 6 : Mise en correspondance d'exécution et conception II

Selon la figure 6, on a deux façons différentes d'arriver à la conception de M1 :

1. Création de *Spéc. Int. M2* et ensuite *de Spéc. Interface* et ensuite *Spéc. Int. M1*
2. Passage direct de *Spéc. Int. M2* à *Spéc. Int. M1*

Il y a bien sûr d'autres cas pratiquement possibles, mais nous ne les considérerons pas parce qu'ou bien ils sont inacceptables (concevoir M1 sans aucune spécification va à l'encontre de toute pratique d'ingénierie) ou parce qu'ils sont moins fréquents et ne changent rien à notre démarche (écriture directe de *Spéc. Interface,* par exemple*)*

Si, une fois *Spéc. Int. M2* écrite*,* les considérations de la section précédente sont valides, quelles sont les implications de l'écriture de *Spéc. Int. M2* sur les connaissances des concepteurs ?

Demandons-nous donc qui a la responsabilité de décrire l'interface de M2. Nous dirons avant tout qui ne devrait pas la décrire. :

1) Le concepteur de M2. Parce que le concepteur de M2 risquerait, malgré lui, d'introduire des détails du fonctionnement interne de sa machine qui d'une part ajouteraient du bruit à la spécification et, de l'autre, risquerait de lier l'interface à la structure interne de M2.
2) Le concepteur de M1. Cela pourrait sembler aller de soi, mais il arrive parfois que des ingénieurs du logiciel soient parachutés dans des domaines qui commencent à être automatisés et qu'ils définissent eux-mêmes les interfaces de M2.

Qui connaît M2 outre son concepteur ? Au moins l'utilisateur qui, dans ce cas-ci, joue le rôle d'expert du domaine. C'est donc un utilisateur de M2, en tant qu'expert du domaine, qui doit spécifier *Spéc. Int. M2*.

À titre d'exemple : si on doit interfacer M1 à un alternateur, ce sera un ingénieur expert en mécanique ou électricité qui décrira les interfaces de l'alternateur en termes de types d'entrées et sorties, de séquences, de temporisations, etc. et non l'ingénieur du logiciel ou un expert en télécommunications !

4.2.2. La machine M2 n'existe pas

Puisque la machine M2 n'existe pas et que nous avons fait l'hypothèse que M1 doit être conçue, les deux machines doivent être conçues en « même temps ». Cela veut dire que les deux machines sont le résultat d'une conception architecturale dans laquelle la machine qui aurait dû mettre en œuvre une fonctionnalité globale a été divisée en deux. Le terme *architecturale* a ici une signification très large et est employé comme simple opposition à fonctionnel. Un bon exemple de conception en même temps, c'est la

modularisation d'un programme, les modules (les classes, si on a une approche objet) étant les machines logicielles.

De quelles connaissances avons-nous besoin pour « diviser en deux » une machine ? Minimalement des fonctions que la machine exécuter et de leurs possible organisation en sous fonctions et les caractéristiques de la machine *qua* machine. Indépendamment du type de machine la division implique, en plus de la connaissance des fonctions et de la technologie, la connaissances des règles de conception, des attributs de qualité et de comment ses attributs peuvent être traités. Si on prend, à titre d'exemple, une machine avec une base de donnée éloignée accessible via le WEB, la première division en une machine physique pour l'utilisateur et une (ou deux) pour le serveur WEB et la base de données est pratiquement dicté par la technologie et donc dépend de l'ingénierie des systèmes. Si on considère la modularisation d'une application dans un PC il est clair que les spécifications des interfaces doivent être créées par l'ingénieur du logiciel.

Pour une analyse plus en détail des différents cas voir la section suivante.

4.2.3. Détail des interfaces en M1

Pour analyser les interfaces, nous imaginerons avoir à concevoir une machine matérielle M1 qui doit dialoguer via une connexion physique quelconque avec une machine M2 du même type.

Nous modélisons M1 comme étant constituée de cinq machines, ce qui nous semble être le minimum nécessaire pour pouvoir considérer tous les types d'interfaces machine-machine.

M1 est constituée de :

1. *Noyau* : machine qui exécute les fonctionnalités de base, c'est-à-dire les fonctions pour lesquelles la machine a été créée et qui ont une certaine autonomie par rapport aux machines qu'on peut connecter. Si M1 est un PC, *Noyau* est l'application qui échange des informations avec une application dans M2 pour pouvoir exécuter les fonctions exigées.
2. *MIU* (Matériel d'Interface avec l'Utilisateur) : machine matérielle pour l'interface avec les dispositifs que l'utilisateur emploie pour interagir avec M1. Il ne s'agit pas, par exemple, du clavier et de l'écran mais des cartes qui s'interfacent au clavier ou à l'écran. En ce qui concerne l'analyse du détail des interfaces internes de M1, MIU pourrait aussi bien être une interface avec *Nature*.
3. *PA* (Protocole d'Application) : machine logicielle qui met en œuvre l'interface « logique » entre les deux machines. Il virtualise le dialogue entre les machines et il cache la structure matérielle et logicielle sous-jacente. Ce que nous avons appelé la couche 8 dans l'exemple d'IEC 61850.
4. *LIMC* (Logiciel d'Interface au Matériel de Connexion) : machine logicielle qui dialogue avec le composant matériel responsable de la communication avec le matériel d'interconnexion. Comme exemple de cette interface, on peut considérer les pilotes des cartes d'entrée/sortie d'un système d'exploitation.
5. MIC (matériel d'Interface pour la Connexion) : machine matérielle qui met en oeuvre la connexion avec le matériel d'interconnexion. Il s'agit du matériel dont une machine a besoin pour échanger des données binaires avec le système d'interconnexion (pratiquement, des cartes d'entrée/sortie).

La figure ci-dessous représente les deux machines physiques M1 et M2 reliées via une connexion physique et un lien virtuel. Les deux machines et les utilisateurs/champs sont insérés dans un domaine d'application. La machine M1 est divisée entre les cinq machines décrites ci-dessus. Dans la figure, les cercles représentent les interfaces : 1 et 6 étant les interfaces physiques vers l'externe, 7 l'interface virtuelle et 2, 3, 4 et 5 les interfaces internes. Le composant *Connexion physique* permet de réaliser l'interconnexion physique des deux machines dont le lien virtuel est mis en œuvre par PA. *Connexion physique* peut être un simple câble tout comme un système constitué à son tour de plusieurs machines (Internet, par exemple).

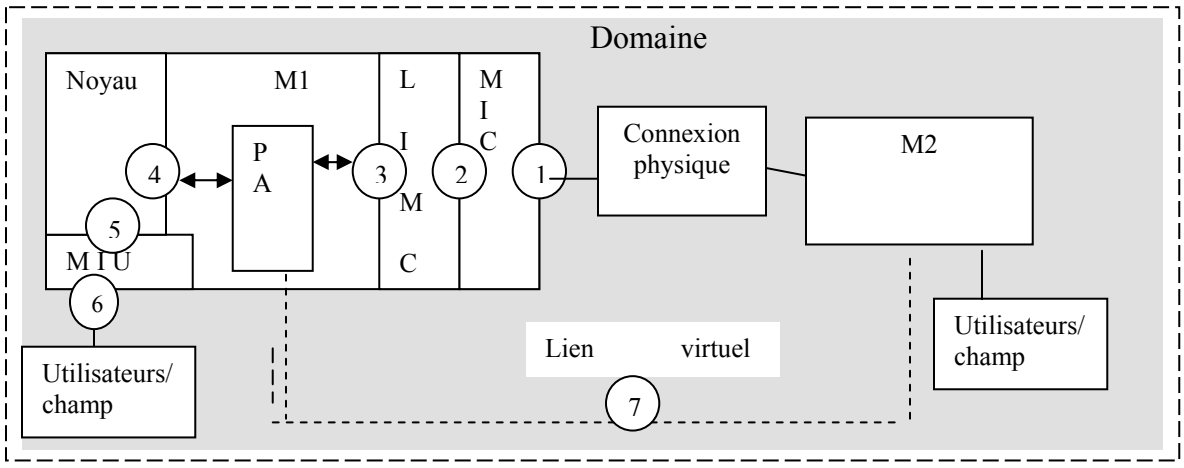

Figure 7 : Interfaces et structure

Après avoir donné cet aperçu de la structure, nous analysons la spécification des interfaces du point de vue des domaines de connaissance.

Interface N° 1

La spécification de cette interface relève de l'ingénierie des systèmes, de l'ingénierie des ordinateurs, de l'électronique et, éventuellement, des télécommunications. Souvent, les MIC sont disponibles sur le marché et les experts des différents domaines ne font que choisir le composant sans écrire une « vraie » spécification. S'il n'y a pas de composant disponible, il s'agit de spécifier une interface qui, probablement, suivra des normes internationales. RS 232-C, USB2 ou IEEE 802.3 sont parmi les exemples les plus connus.

Interface N° 2

Cette interface doit être définie par des ingénieurs des ordinateurs ou, plus généralement, en électronique. Elle dépend de la conception du matériel et elle peut impliquer des ingénieurs du logiciel, si MIC est programmable. Il est clair que si on « ouvre la boîte » MIC, on se retrouve avec une autre machine à laquelle on peut appliquer les mêmes considérations qu'à M1 : on pourra parler de noyau, d'interface matérielle, etc. du MIC. Si l'interface est programmable, bien des choix fonctionnels seront en jeu : le partage entre ce qui est fixé dans MIC et ce qui est à l'extérieur dépend des facteurs technologiques, économiques, de qualité et de l'historique de l'entreprise (compatibilité avec de vieilles versions, par exemple). Si le MIC est un composant acheté, au niveau de l'interface, il suffira de choisir la modalité de fonctionnement à l'aide de paramètres de la carte et le choix des paramètres dépendra de considérations d'ordre systémique. Par rapport aux considérations des sections précédentes, il est fort probable que la machine MIC soit conçue avant LIMC.

Interface N° 3

Dans bien des cas, cette interface est spécifiée par un ensemble d'API de l'environnement choisi. Mais, dans le cas le plus général, on peut la considérer comme le résultat d'une modularisation interne au logiciel qui dépend donc effectivement de connaissances en GL et en informatique car la division implique surtout des considérations de maintenabilité (couplage, lisibilité, etc.).

Interface N° 4

Pour cette interface, nous considérons deux cas :

1. noyau logiciel : interface 4a. Il s'agit de toute évidence d'une spécification qui concerne un partage fonctionnel interne au logiciel et appartient donc au domaine du GL.

2. noyau matériel : interface 4b. Ce cas est équivalent à l'interfaçage à une machine qui existe déjà et donc ne demande pas de connaissances en GL.

Interface N° 5

Cette interface ne nous intéresse que si le noyau est un noyau logiciel ce qui nous ramène à l'interface N 2.

Interface N° 6

En ce qui concerne l'utilisateur, voir la section 5 et, pour l'interface au champ, voir la section 6.

Interface N° 7

Cette interface concerne les échanges d'informations de « haut niveau » entre les deux machines, c'est-à-dire d'informations qui ont une signification dans le domaine d'application. La définition de cette interface peut être très complexe et c'est pour cette raison que des organismes comme ISO ont établi une norme pour l'interconnexion des systèmes ouverts qui divise l'interface en plusieurs couches. Toutes les difficultés de cette interface relèvent du domaine que l'on veut informatiser avec M1 et M2 (comment partager les fonctionnalités) et de la discipline des protocoles (règles pour la sécurité, la fiabilité etc.). Voir à ce propos l'exemple de IEC 61850 et l'annexe A.

Il nous semble important d'ajouter que s'il est vrai que la spécification du lien virtuel entre les machines, comme la spécification des services abstraits entre les couches qui mettent en œuvre les protocoles, ne relèvent pas du GL, il est aussi vrai que la spécification des interfaces des modules qui mettent en œuvre lesdites interfaces appartient au GL.

La table suivante synthétise les considérations de cette section. Les titres des colonnes sont les numéros des interfaces de la figure 7.

	1	**2**	**3**	**4a**	**4b**	**5**	**6**	**7**	**Commentaire**
GL			X	X					
Expertise du domaine	X				X		X	X	
Ing. des systèmes	X				X		X	X	
Ing. des ordinateurs	X	X			X	X	X		
Électronique	X	X			X	X	X		
Télécommunications	X						X		

Table 4 Relation entre interfaces et domaines

En résumé : les connaissances en GL sont nécessaires seulement pour la spécification des interfaces entre les machines logicielles.

4.3. Conclusions

Des considérations présentées dans cette section, nous croyons pouvoir extraire le principe suivant :

> *Seule la spécification des interfaces internes d'une machine logicielle appartient au domaine du GL (P08).*

Le concept de « interface interne » que nous trouvions non important pour catégoriser les interfaces, est ici utile comme élément de délimitation du contexte.

Ce principe implique que ce soit au moment de la conception d'un logiciel (son partage en modules) qu'il faille avoir des connaissances « solides » en GL.

5. Les interfaces personne-machines

Au cours des dernières années, les interfaces personne-machine ont acquis une importance énorme en raison de l'ubiquité des ordinateurs. Cette importance a non seulement modifié les méthodes d'analyse et de conception classiques, mais a donné naissance à un domaine à part entière : le génie de la facilité d'utilisation (*usability*). Nous ne sommes pas sûrs que l'appellation de « génie » soit bonne, car la conception des interfaces personne-machine demande des connaissances et des capacités qui, souvent, sont en conflit non seulement avec celles d'un ingénieur du logiciel mais avec celles de tout ingénieur. Ceux qui, comme nous, croient que l'humain est caractérisé par un haut degré d'imprévisibilité (de liberté, si l'on veut employer un mot très philosophiquement chargé) et que la seule « liberté. » de la machine, ce sont ses pannes, ne peuvent que penser que l'ingénierie est la discipline idéale pour limiter la liberté (les pannes) de la machine mais qu'en même temps elle est terriblement inefficace et incompétente lorsqu'il s'agit de laisser le maximum de liberté et donc de prise de décision à l'humain.

La discipline de la facilité d'utilisation a besoin de connaissances en ergonomie, en psychologie, en arts visuels, etc. et doit être complètement indépendante de la structure du logiciel sous-jacent. L'histoire de l'informatique est pleine d'horreurs à propos d'ingénieurs du logiciel qui ont développé des interfaces personnes-machine. Dans beaucoup d'applications, l'interface personne-machine est un intrant très important pour l'ingénieur du logiciel : un intrant ! c'est tout. Ici, on constate encore une fois la lourdeur de la brève histoire de l'informatique : étant donné que les premières interfaces étaient réalisées par des informaticiens, on continue, par inertie, à les faire réaliser par des informaticiens.

La dernière version du SWEBOK sort, justement, la partie interface personne-machine du génie logiciel et présente une discipline à part : *software ergonomics* (voir encadré1). Le nom de cette discipline nous semble être un non sens : il n'y a rien qui puisse être désigné comme ergonomie du logiciel, car il impossible pour un humain de dialoguer avec un logiciel : un humain a besoin d'une interface matérielle pour pouvoir échanger des informations. Mais, pour essayer de comprendre le choix effectué par SWEBOK d'ajouter logiciel à ergonomie, faisons l'hypothèse que l'expression désigne la communication virtuelle (sans le substrat) entre un humain et une machine. Supposons aussi que cette communication soit organisée en couche, comme le célèbre modèle OSI, et que donc le substrat appartienne à la couche physique. Qu'en est-il des autres couches ? Ne sont-elles pas des couches « immatérielles » qui contribuent à créer un lien personne-machine qui respecte les normes ergonomiques ? Certes. Et SWEBOK nous laisse entendre que si l'on considère une interface réalisée via un écran et un clavier, même si la couleur des champs est générée par du matériel, elle constitue un « choix » logiciel, comme la longueur, comme le type d'interaction, etc. C'est ici que nous ne sommes pas d'accord (à moins de considérer tout comme du logiciel, ce qui nous semble être la source de trop de problèmes). La longueur des champs, la couleur, l'interaction… (ce qui caractérise toutes les couches de l'interface) sont des éléments qui dépendent d'acquis ergonomiques et de certaines nécessités du domaine d'application. Le génie logiciel n'a pas un mot à dire là-dessus. La vraie tâche des informaticiens et des ingénieurs du logiciel est de fournir des outils toujours plus sophistiqués (les machines logicielles qui virtualisent l'interaction) pour permettre la création d'interfaces à des personnes ayant très peu ou même pas du tout de connaissances en informatique.

Ce que nous venons de dire nous permet d'énoncer un dernier principe concernant les interfaces :

> *Une personne ne peut pas échanger des données avec un logiciel* (P09).

Il est clair que l'écriture de la spécification de l'interface personne-machine est ce qu'il y a de plus loin de l'ingénierie du logiciel. Ajoutons que laisser les ingénieurs du logiciel s'intéresser aux interfaces personne-machine conduit, avec une probabilité proche de la certitude, à des projets catastrophiques

6. Interfaces personne-personne

Nous ne considérons les interfaces personne-personne que dans le cadre d'un projet de spécification, de conception et de réalisation d'une machine logicielle ou d'une machine matérielle qui contient du logiciel.

Rappelons que, selon le principe P08, l'échange d'information via l'interface personne-personne nécessite, comme tous les échanges, un substrat. Nous caractérisons ce substrat en fonction de sa façon de garder l'information. Nous parlerons donc d'échanges volatiles et d'échanges rémanents. Échanges volatiles par rapport aux substrats externes et non par rapport à la mémoire « interne » des humains.

Échanges volatiles

Nous appelons volatiles les échanges qui ne sont pas mémorisés à l'extérieur du cerveau des intervenants. La majorité des échanges volatiles sont des échanges vocaux qui deviennent rémanents s'ils sont enregistrés[14]. Même si les échanges volatiles sont essentiels et conditionnent la qualité des processus et des produits, nous ne les considérons pas, d'une part parce qu'ils nous éloigneraient trop de notre propos et de l'autre parce que nous n'avons pas les compétences nécessaires. Surtout si l'on considère que ces échanges deviennent particulièrement complexes lorsque deux personnes ont besoin d'une troisième pour pouvoir communiquer entre elles, comme c'est le cas dans tous le projets d'une certaine envergure : équipes d'au moins trois ou quatre personnes et durée d'au moins quelques mois. La troisième personne se transforme en « substrat » intelligent qui peut modifier l'information échangée de manière imprévisible.

Échanges rémanents

Dans cette catégorie, nous ne considérons pas les enregistrements vocaux, non seulement parce que leur emploi est très marginal, mais parce qu'avec les nouvelles technologies on arrivera bientôt à des transcriptions acceptables, ce qui les fera entrer dans la catégorie de la documentation écrite « classique ». Nous considérons donc les échanges dont le substrat est l'écriture sur papier ou électronique : ce que l'on appelle la documentation de projet, que nous considérons sans tenir compte des caractéristiques du support physique[15].

Nous allons donc faire quelques considérations sur la documentation. Rappelons avant tout la définition de logiciel de [1] : « Les *programmes, les procédures et la documentation associée et les données qui concernent le fonctionnement d'un système informatique »* qui considère la documentation comme partie intégrante du logiciel. Nous dirions même plus : la documentation a toujours eu une place spéciale dans le génie logiciel : dans les débuts de l'informatique parce qu'elle était pratiquement absente (on[16] écrivait du code et, dès que quelque chose bougeait, on disait que c'était fini), ensuite parce que, dans certaines organisations, elle a pris une importance excessive. Qu'il suffise de rappeler que la norme IEEE 12207-1 prévoit 84 types différents de documents dans un projet dont 32 types pour la qualité et 29 pour la gestion.

Il est clair que, devant tous ces documents, on peut avoir l'impression d'être envahis par le papier et que le programme exécutable, le but final de l'automatisation, est un peu laissé pour compte. Des réactions très radicales d'opposition au génie logiciel « classique » comme *Extreme programming,* trouve dans cette panoplie de documents leur meilleure justification.

Cette quantité « excessive » de type de documents en Amérique et en Europe est due aussi à l'instabilité du personnel qui change facilement d'employeur. La documentation devient alors une composante importante de ce que l'on appelle, improprement, la connaissance de l'entreprise.

Analyser les interfaces personne-personne dans l'optique que nous proposons équivaut donc à une analyse de la documentation : des types de documents, de leur quantité, du moment de leur production, des auteurs, etc. Mais tout cela relève-t-il d'une façon quelconque du génie logiciel ? Nous croyons que non, que ce sont les connaissances en ergonomie et en sciences cognitives qui servent à établir les standards de documentation que les ingénieurs du logiciel doivent suivre dans leurs documents. Nous insistons :

---

[14] En ce sens, le proverbe latin *Verba volant scripta manent* aurait besoin d'une mise à jour.

[15] Si le support physique est une machine électronique (documentation hypermédia), l'efficacité de l'échange personne-personne dépend d'une interface personne-machine qui s'interpose entre les deux, l'interface qui permet d'accéder à la documentation. Il suffit de considérer une documentation avec des hyperliens mal conçus pour voir comment l'interface personne-machine peut causer des pertes de temps et de mauvaises compréhensions. Nous ne considérons pas cette interface personne-machine parce que nous sommes de simples utilisateurs et non des concepteurs. Par rapport à notre figure 4, nous sommes en dessous de la ligne.

[16] Ce « on » n'englobe pas des gens comme Dijkstra, Hoar et beaucoup d'autres universitaires qui avaient besoin de la documentation pour enseigner et pour expliquer leurs travaux de recherche. Ce « on » fait référence à la majorité de ceux qui travaillaient dans les industries qui commençaient à naître et qui se lançaient avec une naïveté redoutable dans le nouveau « business ». Avec naïveté et, bien souvent, avec une ignorance tout aussi redoutable.

« leurs » car, parmi les 84 types de la norme IEEE 12207-1, il y en a un très grand nombre qui ne les concerne pas.

7. Interfaces machine-nature

Comme pour les interfaces personne-machine, la machine des interfaces nature-machine doit être physique.

La figure suivante montre une chaîne d'acquisitions, simplifiée et unidirectionnelle, de la mesure d'un phénomène appartenant à ce que nous avons appelé *Nature*.

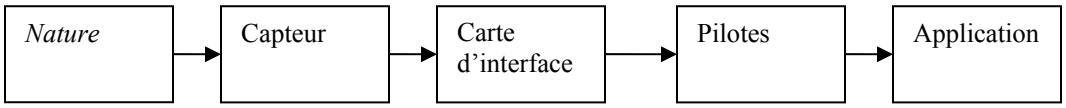

Figure 8 : Chaîne d'acquisitions

L'interface machine-nature se réduit à l'interface avec le capteur, toutes les autres interfaces étant des interfaces machine-machine. Si le capteur est un capteur « intelligent », c'est-à-dire doté d'une unité de traitement et de programmes, il peut être structuré en machines comme nous l'avons fait pour M1 dans la figure 7. Ce qui est important dans l'interface machine-nature se situe alors là où le phénomène naturel est transformé en une entité concrète et manipulable qui le représente. Si, par exemple, l'application doit traiter une température, le capteur aura minimalement une substance qui change en fonction de la température (métal qui se dilate). Si le capteur est « intelligent » et donne, sur un lien RS 232-C, une température en unités d'ingénierie, cela ne change rien à l'interface : ce qui change, ce sont les interfaces internes de la machine globale.

En fonction de ce que nous venons d'exposer, il est clair que la spécification de l'interface machine-nature est du ressort de l'expert du phénomène que l'on veut mesurer et de l'expert en électronique. De ce type d'interface, l'ingénieur du logiciel est complètement absent.

8. Conclusion

Dans la table suivante, pour chaque catégorie d'interfaces, nous indiquons si elle est concernée par l'automatisation et par le génie logiciel tel que nous l'envisageons.

Interface		**Automatisation**	**GL**	**Remarque**
Personne-Machine		Oui	Non	Les théoriciens sont d'accord mais, en pratique, les ingénieurs du logiciel continuent à faire des dégâts.
Personne-Nature		Non	Non	Comme domaine d'étude préalable à l'automatisation seulement.
Personne-Personne		Oui	Oui	Parce qu'elle influences les approches à l'automatisation et en particulier la documentation.
Nature-Machine		Oui	Non	Pour le matériel seulement.
Nature-Nature		Non	Non	Comme domaine d'étude préalable à l'automatisation seulement.
Machine-Machine				
	Matériel-Matériel	Oui	Non	Pour les connaissances personnelles.
	Matériel-Logiciel	Oui	Non	Matériel « prioritaire ».
	Logiciel-Logiciel	Oui	Oui	Avec les bémols présentés en 4.2.3..

Table 5 : Synthèse des

Même si la table 5 ne fait référence qu'aux spécifications des interfaces, il possible et envisageable d'extrapoler à beaucoup d'autres artefacts des processus d'automation. Ce qui pourrait faire penser que nous sommes en train de restreindre le génie logiciel aux activités de conception et de codage, ce qui pourrait être considéré comme un retour à la situation qui prévalait avant que l'on commence à parler de génie logiciel.

Nous croyons qu'il y a effectivement un « retour du même » mais, comme toujours, l'histoire, et en l'occurrence l'histoire de l'automatisation, a laissé sa trace et le « même »… n'est plus le même. Dans le cas du génie logiciel, l'histoire nous a montré qu'il fallait que d'autres expertises entrent en jeu et aient un poids réel dans les projets ; que les processus d'automatisation ont souvent une complexité telle que les domaines d'applications doivent reprendre la centralité qui est la leur et qui a été « volée » par certain programmeurs que la puissance de l'ordinateur a aveuglés.

Annexe A : Protocole de communication

Mise en contexte

Une petite entreprise qui opère dans le contrôle des procédés et qui construit des sous-systèmes ayant des fonctions primaires très spécifiques et qui doivent communiquer avec d'autres sous systèmes. En raison de contraintes matérielles, la communication doit être de type *start-stop*. L'entreprise choisit de développer une nouvelle couche liaison de données.

Voici une liste d'activités. Nous ne considérons pas les activités qui précèdent la définition du protocole.

Activités

1. Définition du protocole
    - Mode de fonctionnement (*fully balanced*).
    - Format du LPDU.
    - Définition des codes de fonction.
    - Définition des méthodes de récupération.
    - Définition abstraite des SAP et LSDU.
    - ...
2. Prototype pour valider les idées et tester en gros les fonctionnalités.
3. Finalisation de la spécification.
4. Ingéniérisation du produit.
5. Tests d'acceptation.
6. Maintenance[17] (améliorations, corrections etc.).

Nous analysons chaque activité en faisant des considérations par rapport aux connaissances nécessaires pour compléter l'activité de manière optimale.

Spécification du protocole

Activité qui demande seulement des experts du domaine. Les experts choisiront le langage de description qu'ils croient le plus apte à décrire le protocole et le plus proche de leur domaine. Aucune connaissance en GL n'est requise.

Prototype

Cette activité ne demande pas de connaissances en GL, mais simplement une capacité à aligner une instruction après l'autre, comme au bon vieux temps. Il s'agira sans doute d'un prototype jetable réalisé par des « protocolistes »

Finalisation de la spécification

Complètement pris en charge par des experts du domaine

Ingéniérie du produit

[17] Nous considérons improprement la maintenance comme une activité.

C'est ici que le GL devient important. On pourrait dire qu'il s'agit là d'une pétition de principe, car, si l'on parle d'ingénierie après avoir développé une partie du produit, il est clair que cette ingénierie sera le GL. Nous croyons qu'il ne s'agit pas d'une pétition de principe, car les activités qui précèdent ce que nous appelons ingénierie ne sont pas des activités « non scientifiques » ou « non techniques » ou « floues », mais ce sont des activités systématiques au même titre que le GL mais, si on veut, dans un autre génie[18] : celui des protocoles. Pour nous, « ingéniérisation du produit » signifie *application des connaissances du GL* pour construire un produit logiciel de manière maîtrisée et de manière à ce qu'il soit maintenable avec des coûts raisonnables.

Test d'acceptation
Aucune connaissance en GL.

Documentation utilisateur
Aucune connaissance en GL.

Maintenance
Toute la maintenance demande une connaissance très grande du GL avec bien sûr des connaissances du domaine des protocoles (pas nécessairement exercées par les mêmes personnes).

Exemples de choix faits en se fondant sur les connaissances du domaine

- Délimitation des trames.
- Adressage.
- Structure des octets définissant les codes de fonction.
- Type de contrôle d'erreur.
- Nombre de retransmissions.
- Synchronisation entre trames.
- Prévention des *Deadlocks* (pour le protocole).
- ...

Exemples de choix faits en se fondant sur les connaissances du GL

- Nombre de *threads* (fils d'exécution).
- Type de dialogue entre la couche trois et la deux (synchrone ou asynchrone).
- Synchronisation entre les *threads*.
- *Deadlocks* (pour les *threads*).
- Paramétrage de configuration.
- …

Conclusions
La séparation entre les connaissances des deux domaines nous semble très claire. Pratiquement seulement deux « activités » concernent des connaissances du GL (*Ingéniérisation* et *maintenance*). Il est intéressant de considérer que des éléments comme *Deadlock* et *synchronisation* apparaissent dans les deux domaines, ce qui nous semble indiquer que, même quand deux domaines[19] sont concernés par les « mêmes concepts », la façon de les traiter est différente. En effet, on peut parler de synchronisation dans bien des domaines (même en psychologie), sans que cela veuille dire qu'il s'agisse de GL ou de protocoles de communication entre machines.

Software ergonomics

[18] Le fait de caractériser comme de l'ingénierie tout ce qui est systématique (comme ingénierie des exigences, par exemple), nous semble excessivement réducteur. Nous croyons que non seulement des philosophes comme Kant ou Hegel ont été très systématiques mais même des poètes comme Valéry auraient beaucoup de choses à enseigner à des ingénieurs en ce qui concerne la systématicité, la logique, la maîtrise, etc. Comme me l'a fait noter le professeur L. Martin, l'ingénierie est en train de faire ce que la philosophie faisait au moyen âge. Cette considération est une transposition très appropriée des idées d'Heidegger sur la technique comme fin de la métaphysique.

[19] Le fait de mettre les *deadlocks* dans un secteur de l'informatique, comme le propose SWEBOK, ne fait que renforcer notre position !

L'ergonomie du logiciel est l'une des huit (8) disciplines reliées au génie logiciel selon SWEBOK. Pour l'ergonomie SWEBOK cite la définition de Comité technique 159 d'ISO : « *Ergonomics or (human factors) is the scientific discipline concerned with the understanding of the interactions among human and other elements of a system, and the profession that applies theory, principles, data and methods to design in order to optimize human well-being and overall system performances* ».
La traduction française de *Software ergonomics* par *ergonomie du logiciel* nous permet de voir que la duplicité du « du » (« du » objectif ou subjectif) crée deux significations complètement différentes :

1) l'ergonomie par rapport aux machines qui exécutent le logiciel ;
2) l'ergonomie du logiciel lui-même.

Le choix de SWEBOK est clairement le 1). Dans le corps de l'article nous croyons d'avoir montré que cette signification ne peut pas être retenue. Le choix de SWEBOK de mettre l'ergonomie (logicielle) comme une discipline reliée au génie logiciel est un bon choix… si on laisse tomber le mot *logiciel* car se mot ne fait qu'obscurcir les concepts. Le choix de ne pas ajouter *logiciel* est loin d'être anodin, surtout dans un domaine où bien des gens confondent encore la conception du logiciel avec celle des interfaces personne-machine !

Nous croyons que la deuxième signification est très intéressante pour le génie logiciel. Elle est important surtout par rapport à la documentation qui fait partie intégrante de la définition de logiciel selon IEEE [1]. Pourquoi cette deuxième signification d'ergonomie du logiciel est-elle si importante ? Parce que la documentation est le point central (et faible) du génie logiciel et parce que l'étude des caractéristiques de la documentation et son amélioration demande des connaissances très lointaines de celles nécessaires pour bâtir des programmes.
L'ergonomie du logiciel concerne l'amélioration des performances du projet plutôt que celles du « système » final comme l'ergonomie considéré en 1). L'ergonomie du logiciel serait l'ergonomie appliquée aux outils et à la documentation employés dans les projets contenant du logiciel. Serait-elle une discipline externe ou ferait-elle partie du génie logiciel ? Nous n'avons pas de réponse : avant de répondre à cette question, il faut encore beaucoup de travail de déblayage.

Encadré 1

Références

[1] IEEE Std 610.12-1990, *IEEE standard Glossary of Software Engineering Terminology*.

[2] IEEE Std 830.12-1998, *IEEE Recommended Practice for Software Requirement Specification*.

[3] Ivan Maffezzini, Alice Premiana, Bernardo Ventimiglia, *Prolégomènes à une critique du génie logiciel, Partie I : contextualisation*, in Génie Logiciel, Septembre 2003, No 66.

[4] IEEE, *SWEBOK,* 2004 version, http://www.swebok.org/