MISSION CRITICAL ti

Ontologie pour l'entreprise digitale

De la complexité du legacy à une plate-forme digitale agile

1. Défi digital, défi logiciel

Le matériel informatique (serveurs, stockage, réseau, *cloud*, tablettes, *smartphones*), toujours plus performant et plus fiable, est disponible à des prix qui n'arrêtent pas de décroître.

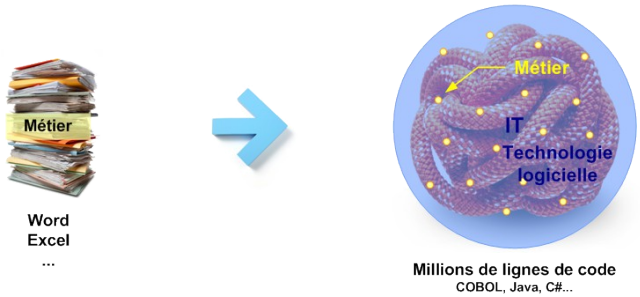
Il en va tout autrement du logiciel *mission-critical* d'entreprise, développé durant des décennies sur *mainframe*, en COBOL ou PL/1. Son maintien en conditions opérationnelles consomme jusqu'à 85% du budget informatique qui, crise oblige, est en diminution ; le passage à des langages modernes, Java ou C#, ne change pas fondamentalement cette situation.

Innover dans ces conditions est très difficile, alors que les demandes du métier se font pressantes sur de multiples fronts : agilité, vision unique des clients, développement de ventes croisées, exploitation des données des réseaux sociaux, relation digitale avec les clients et partenaires d'une qualité égale à celle des grandes applications du web, etc.

Le **défi logiciel** à relever est majeur : il faut réduire drastiquement le TCO (*Total Cost of Ownership*) des applications critiques[1] tout en augmentant considérablement leur flexibilité et leur qualité[2].

1.1 Origine de ces difficultés

Dans la pratique actuelle (qu'elle soit de type *waterfall* ou agile), les informaticiens codent dans des programmes à la fois de la **connaissance métier** (exprimée de manière souvent ambiguë et incomplète) et de la **technologie logicielle**. Ce qui est propre au **métier** est **noyé** dans de nombreuses lignes de code **technique** :



Premier défi : le « quoi »

Comme l'écrit Fred Brooks, l'auteur du fameux *The Mythical Man-Month*[3], "*The hardest single part of building a software system is deciding precisely what to build*".

Les responsables métier ont en général beaucoup de difficultés à formuler leurs exigences avec la précision et l'exhaustivité que les informaticiens requièrent. Et lorsqu'ils innovent, ils sont dans un processus de découverte qui rend l'expression des besoins inévitablement changeante.

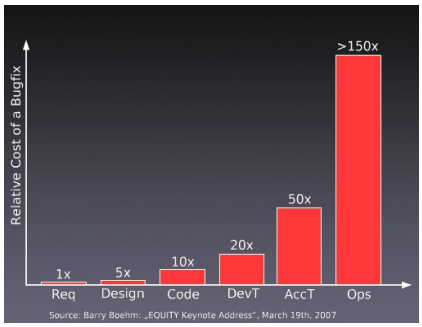
[1] En opposition aux applications logistiques telles l'ERP et aux logiciels de productivité (Office...)
[2] Voir par exemple http://www.cio.com/article/564613/5_Reasons_Businesses_Still_Hate_Enterprise_Software
[3] http://en.wikipedia.org/wiki/The_Mythical_Man-Month

Deuxième défi : le « comment »

Le développement de programmes est une activité qui est complexe et coûteuse : les budgets d'investissement et de maintenance sont très élevés, les délais presque toujours dépassés, la qualité insuffisante et la tolérance au changement problématique, toutes difficultés que les études CHAOS du *Standish Group* et de nombreux articles illustrent bien :

Troisième défi : le « quoi » et le « comment » imbriqués

Quand ces deux aspects sont imbriqués – les informaticiens codent de la connaissance métier avec un langage de programmation (COBOL, PL/1, Java, C#, Python...) – la complexité (essentielle) du **problème** et la complexité (accidentelle) de la **solution** technique se conjuguent : le programmeur doit maîtriser une **complexité au carré**. Comme le problème reste mal défini, des changements au niveau métier affectent de très nombreuses lignes de code avec des effets de bords imprévisibles qui nécessitent d'importants efforts de correction. Plus les changements sont nombreux, plus la **traînée** que forme le code déjà écrit augmente, ce qui explique qu'une erreur identifiée durant les tests d'acceptation est **50 fois** plus coûteuse que la même erreur identifiée lors de la spécification :



1.2 Comment relever ces défis ?

En premier lieu, il faut séparer la **spécification** (la définition du problème) de l'**implémentation** (sa solution informatique) pour :

- maîtriser **indépendamment** la complexité du problème et la complexité de solution technique et éviter ainsi l'effet multiplicatif (la complexité « au carré » ;
- **utiliser des langages différents** : adaptés à l'expression de la connaissance métier pour le « quoi » et adaptés à la programmation d'un ordinateur pour le « comment ».

Le langage de description du problème doit être **déclaratif**, c'est-à-dire n'exprimer que le « quoi » et surtout pas le « comment » (ce que font les langages de programmation qui sont **impératifs**).

La **spécification** ne doit pas être qu'une documentation : exprimant l'essence du problème métier, elle **doit piloter l'application** ; le programme et la spécification sont alors toujours alignés.

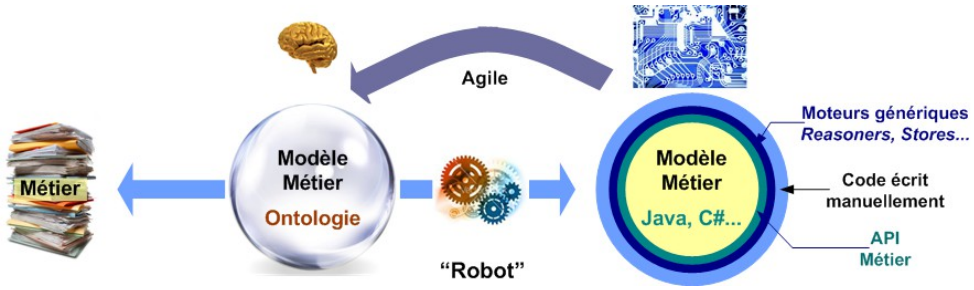
Enfin, comme pour les processus industriels modernes, il faut **automatiser** la chaîne de production d'une application, pour réduire les coûts et les délais et augmenter la qualité et la flexibilité du logiciel.

1.3 Approche ontologique de MCit

MCit a développé toute la technologie qui permet la mise en œuvre de ces principes. Cette nouvelle ingénierie du logiciel s'appuie sur des concepts très innovants, au cœur des travaux du **W3C** (*World Wide Web Consortium*) sur le *Semantic Web*[4]. Elle se structure autour des principes suivants.

[4] http://www.w3.org/standards/semanticweb/

- Le recours à la **science** – théories, modèles et mathématique adéquate (la logique formelle) – en passant de l'artisanat actuel (construire un logiciel comme les romains construisaient un pont) à une démarche scientifique (« calculer » un logiciel lors de sa conception comme on calcule un pont avant de le construire).
- Modéliser le problème métier en créant une **ontologie**[5] : la représentation précise, concise et exécutable de **100% du « quoi »** (concepts, règles, processus et données décrivant toute la logique métier), **indépendamment du « comment »** informatique.
- Pouvoir **tester**, **expliquer**, **vérifier et valider** le modèle avant d'écrire la première ligne du programme associé, parce que les spécifications sont exécutables.
- Réduire fortement les coûts de développement grâce à des **« robots »** logiciels qui vont produire automatiquement l'équivalent de l'ontologie dans le langage de programmation choisi par l'entreprise (industrialisation et non codage « à la main »).
- Les informaticiens utilisent le modèle métier grâce à une **API**[6] **métier**. Sur cette base, ils développent ce qui n'est pas dans le modèle (et qui ne doit pas y être pour ne pas polluer le « quoi » par le « comment ») – pour les aspects IHM, BD, SOA[7], middleware[8], intégration... – avec les **technologies**, les **outils** et l'**architecture** de **leur choix**.
- Les informaticiens disposent d'une **plate-forme logicielle générique** qui regroupe tous les composants (indépendants de la logique applicative) requis par une application. Cette plate-forme – comme un lecteur de DVD « jouant » l'ontologie – abaisse fortement le coût de production et augmente la fiabilité et la performance de l'application, en réutilisant des composants standards (*reasoners*, *stores*, *process engine*, etc.) optimisés et ouverts sur le futur (exploitation automatique du parallélisme des processeurs multi-coeurs).
- L'application **peut facilement être modifiée** : l'ontologie est adaptée en fonction des nouvelles exigences et l'application est **générée à nouveau**. La **qualité** découle du continuum logique qui s'étend de l'ontologie au programme piloté par cette ontologie.
- L'application est développée de manière **agile** à partir de spécifications qui commencent par être incomplètes. À chaque itération, l'ontologie est enrichie, validée par les experts métier et l'application est générée. Le passage de l'idée au programme est très **rapide**.
- Comme l'ontologie est indépendante d'un langage de programmation, il est possible de **réutiliser** la même **ontologie** avec une **technologie** informatique **différente**.



1.4 ODASE

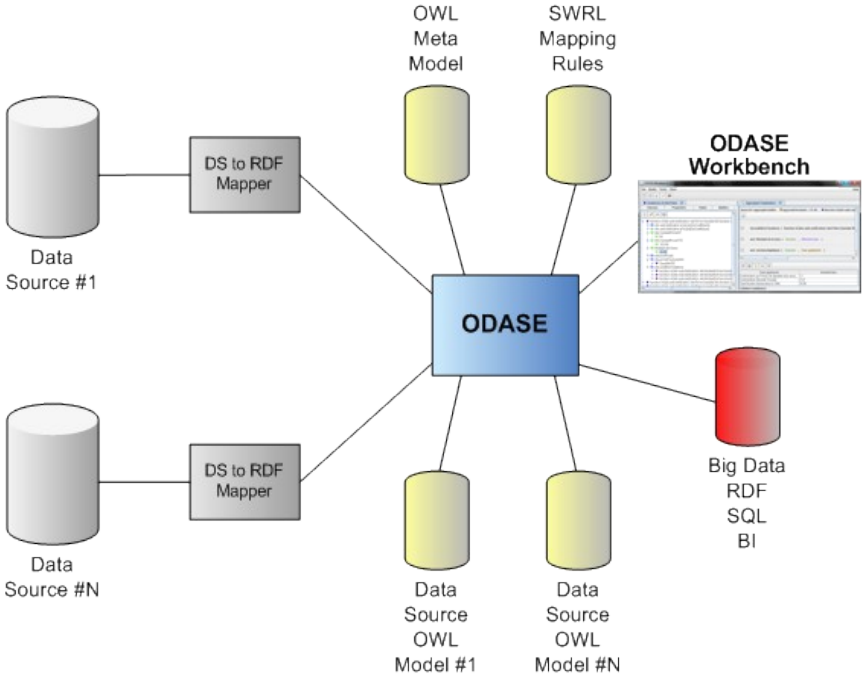
ODASE™ (*Ontology-Driven Architecture and Service Engineering*), est le nom donné par MCit aux outils et à la plate-forme qui exploitent tous les bénéfices des ontologies pour produire des applications flexibles, de qualité, moins chères et plus rapidement.

Un point essentiel est que les langages utilisés au niveau de la modélisation ontologique sont ceux définis par le W3C Semantic Web[9] garantie de pérennité.

[5] http://fr.wikipedia.org/wiki/Ontologie_(informatique)
[6] Application Programming Interface
[7] Interface Homme Machine, Base de Données, *Service Oriented Architecture*
[8] http://en.wikipedia.org/wiki/Middleware_(distributed_applications)
[9] http://www.w3.org/standards/semanticweb/

1.5 ODASE et Big Data

Les technologies du web sémantiques sont au cœur d'ODASE. Elles apportent l'intelligence requise pour pouvoir exploiter des **données extérieures** (comme l'*Open Data*) et avoir une vue transversale et intégrée sur des données en **silo**, de manière opérationnelle (par exemple « vue unique du client ») ou pour la *Business Intelligence* :



2. Produits et services de MCit

- ODASE, plate-forme et outils
- Support et maintenance de ODASE
- Services de développement et d'assistance ontologique
- Services de développement et d'assistance ODASE
- Études : modernisation, architecture métier, projet pilote, BigData sémantique...

3. Quelques-unes des questions fréquemment posées

Q – Le développement d'une ontologie est-il à la portée d'un informaticien d'entreprise ?

Oui. Les programmeurs COBOL font de bons analystes métier capables de construire une ontologie avec les experts du domaine. L'expérience montre que le passage de COBOL à l'ontologie est beaucoup plus facile que le passage de COBOL à Java qui, sur le plan métier, n'apporte aucun bénéfice.

Q – Ne déplace-t-on pas la complexité du développement de l'implémentation à la modélisation ?

Non. La modélisation fournit une ontologie très compacte, précise et formelle qui peut être testée et vérifiée. La recherche d'erreurs éventuelles porte sur peu d'objets de connaissance (concepts, propriétés, axiomes, règles) qui ont entre eux des relations logiques claires et qui sont exprimés avec un vocabulaire métier.

Q – Comment se répartissent les efforts entre l'ontologie et la programmation ?

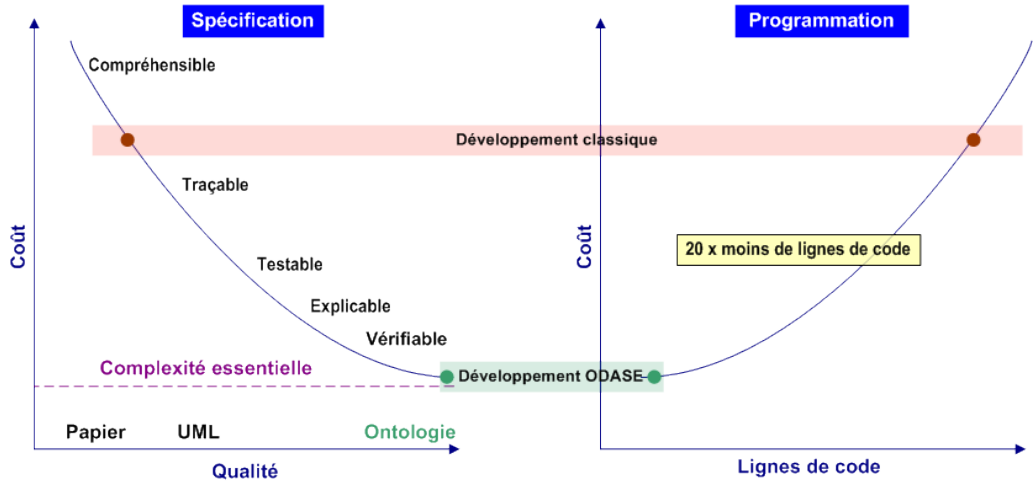
Environ 50% de l'effort total est consacré au problème métier, la construction et à la validation de l'ontologie, et 50% à l'implémentation. Le nombre de lignes de code (ontologie et programme) est réduit d'un facteur 20 par rapport à un développement classique, ce qui a un impact énorme sur le prix, la qualité, les délais et la maintenance.

Q – Peut-on mettre en œuvre cette approche de manière progressive ?

Oui et il convient de le faire. « *Un système complexe qui fonctionne est invariablement le résultat d'une évolution à partir d'un système simple qui fonctionnait. Un système conçu au départ avec toute sa complexité ne fonctionne jamais et ne peut pas être modifié pour fonctionner.* » (John Gall, *Systemantics*). L'approche permet de procéder par extension et spécialisation de l'ontologie.

Q – Quelle est la source d'une réduction du coût aussi importante ?

Elle provient de la combinaison (1) d'une spécification complète et vérifiable du métier. (2) de la génération automatique d'un programmme conforme à la spécification et (3) d'une réduction très importante du nombre de lignes de code à écrire manuellement :



4. En conclusion

La séparation claire entre la définition du problème (l'ontologie) et l'implémentation (les programmes), couplée à la génération de code et au pilotage du programme par l'ontologie offre les **bénéfices** majeurs suivants :

- un coût considérablement réduit pour le développement et la maintenance
- le passage rapide de l'idée (dans l'ontologie) à la réalisation (dans le programme)
- la possibilité d'expliquer, vérifier et valider la définition métier avant la programmation
- une prise en compte rapide des changements métier sans risques de déstabiliser le code
- une répartition plus équilibrée des tâches entre métier et informatique
- un point unique de définition du métier partagé par le métier et l'IT
- une définition métier toujours à jour puisqu'elle pilote les applications
- la connaissance métier est explicite : elle n'est pas perdue dans des programmes
- la complexité est maîtrisée : quelques centaines de concepts, règles et processus
- les informaticiens peuvent s'appuyer sur la sémantique non ambiguë de l'ontologie
- le code à écrire manuellement est considérablement réduit, d'au moins 20 fois
- les changements dans l'ontologie sont immédiatement pris en compte par l'application
- la connaissance métier est formalisée indépendamment de tout langage informatique
- cette connaissance est réutilisable avec d'autres technologies, sans devoir repartir de zéro
- cette approche s'appuie sur des standards ouverts (W3C) basés sur des technologies web
- le *Big Data* devient sémantique (données, informations et connaissances)
- cette démarche et technologie peut être utilisée sur site ou dans le *cloud*

Pour toute information complémentaire : info@missioncriticalit.com