« Livre blanc des fondements de la modélisation système : modèle de données et architecting »

Vous avez compris l'intérêt de mettre en place l'ingénierie système pour améliorer la qualité de vos produits ou en référence à des normes de sûreté de fonctionnement ou de cyber sécurité ou d'analyse de la valeur ou d'autres attributs comme RSE, ou encore pour répondre à des exigences de maturité des processus (CMMi, SPICE, Automotive-SPICE).
Vous avez peut-être déjà mis en œuvre une démarche de gestion des exigences mais vous êtes conscient que c'est insuffisant.

- Vous voulez réaliser des modélisation systèmes au juste nécessaire et en même temps complètes et cohérentes ?
- Vous souhaitez modélisez vos exigences et vos architectures de manière intégrée et cohérente ?
- Vous voulez que vos équipes opérationnelles soient capables de prendre en main en quelques jours et de manière efficace un outil de modélisation parfaitement adapté à vos besoins ?

Nous avons une réponse opérationnelle et éprouvée à vous proposer.

Voici notre manifeste pour expliquer pourquoi notre technologie arKItect est pertinente et unique.

Manifeste pour une modélisation système (MBSE) simple d'utilisation et efficace

- ***Des types et des attributs au juste nécessaire et pas un langage commun pour tous***
- ***Des messages et pas des ports***
- ***Des structures hiérarchiques explicites et pas une abstraction des structures de données***
- ***Des points de vue cohérents plutôt qu'une seule hiérarchie complexe***
- ***Des chaines déduites de l'architecture et pas une multitude de diagrammes***
- ***Un langage formel bien-fondé et pas de concepts complexes sans traduction logique***

Bénéfices attendus :

- **Une formation rapide des équipes en quelques jours grâce à un formalisme intuitif basé sur peu de règles**
- **Une meilleure compréhension de l'ingénierie système par l'explicitation des points de vue**
- **Une utilisation bien plus efficace car le modèle est de taille minimale par rapport au système à modéliser**
- **Une meilleure interopérabilité car il n'y a pas de complexité cachée dans le format de stockage**

Nos utilisateurs rapportent des gains de productivité de plus de 40% par rapport à d'autres solutions, voire beaucoup plus.

Retrouvez l'explication sur chaque principe dans les pages suivantes :

Des types et des attributs au juste langage commun pour tous

Parce que les systèmes sont tous différents par leur taille, leur criticité, leur dynamique, leurs domaines métiers, … Et qu'on préfère un petit manuel que tous les acteurs métiers comprennent plutôt qu'un gros manuel sensé répondre à tout dans lequel 80% des choses ne serviront jamais à rien et où au contraire il manque des choses importantes que des acteurs métiers utilisent tout le temps.

Des messages et pas toujours des ports

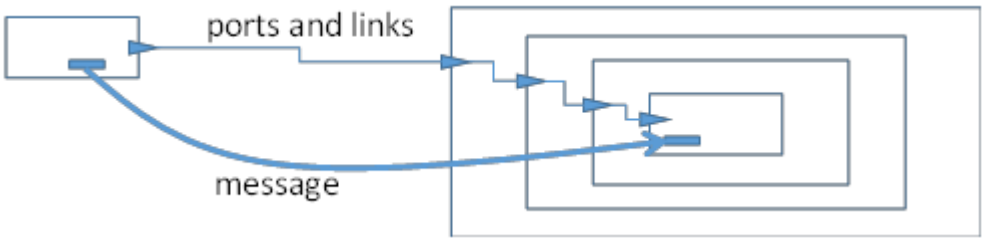
Nous donnons la parole au maître, Alan Kay (l'un des fondateurs de Smalltalk et donc de l'orienté objet :

*Internet inspiration: Neither the computers on the Internet, nor their operating systems, nor the objects in their software systems have to be identical in order for the system to work. Instead, the heterogeneous mixture that the system is made from must simply obey message passing conventions in order to interoperate. We can note in passing that one of the biggest problems in the development of object-oriented SW architectures, particularly in the last 25 years, has been an enormous over-focus on objects and an under-focus on messaging (most so-called objectoriented languages don't really use the looser coupling of messaging, but instead use the much tighter "gear meshing" of procedure calls – t**his hurts scalability and interoperability**).*

Extrait "Proposal to NSF - Granted on August 31st 2006, Steps Toward The Reinvention of Programming", Alan Kay, Dan Ingalis, Yoshiki Oshima, Ian Piumarta, Andreas Raab

Les ports sont la traduction graphique des appels procéduraux.

En termes graphiques, voilà la différence :



- *Un message ici est un seul objet avec deux instances dans le modèle (une en production et une en consommation. Pour obtenir la même chose avec des ports et des liens, il faut 5 ports et 4 liens, donc autant d'objets différents.*
- *C'est à multiplier par le nombre d'interactions dans le système (messages) !*
- *C'est à multiplier par le nombre de consommateurs différents de chaque message !*
- *C'est encore plus pénalisant en cas de modification de l'architecture, les ports et liens étant à supprimer puis recréer si l'on souhaite repositionner un bloc dans l'architecture. Cela rend la conception hyperstatique car les utilisateurs vont préférer ne rien toucher que de faire les modifications qu'ils souhaiteraient, compte tenu de la masse des modifications à réaliser lorsqu'on réorganise des blocs.*
- *Tous ces objets inutiles vont se retrouver partout : stockage, format d'échange, configurations, taille du modèle.*

Nota Bene

1. *Les messages peuvent être utilisés autant pour les interfaces fonctionnelles que systèmes ou physiques*
2. *Dans le domaine logiciel, les ports peuvent être très utiles, ils sont à la base de la "virtualisation" et de la “conteneurisation” :*
3. *Le logiciel par l'indépendance qu'il a pu développer avec le hardware, c’est-à-dire avec les ressources qui le réalisent, justifie parfois une approche de modélisation différente.*
4. *Il n'y a aucune contradiction à intégrer à la fois les messages et les ports. C'est possible dans notre formalisme. Les ports sont pertinents pour caractériser des librairies ou des composants génériques dont les interfaces sont des invariants. Par exemple avec un logiciel comme Matlab/Simulink, il est nécessaire de définir des ports sur les interfaces des blocks standards et de manière générale pour une librairie ou des composants dont on a décidé à l'avance que leurs interfaces ne seraient pas modifiées.*

Des structures hiérarchiques explicites et pas une abstraction des structures de données

L'orienté objet est modulaire car il fait l'abstraction des structures de données. C'est absolument parfait pour les logiciels dans la plupart des cas. Mais le monde autour de nous est constitué en strates hiérarchiques stables dans le temps comme l'a expliqué Herbert Simon dans son article "The architecture of Complexity" 1962.

Deux choses importantes que tout à chacun pourra constater :

Tout d'abord la stabilité : si l'on s'intéresse à la physiologie d'un être humain, on constate qu'il est composé de systèmes respiratoire, nerveux, sensoriel, musculo-squelettique, digestif…. Il ne semble pas que cela va changer de sitôt. Dans un véhicule, vous avez un moteur et un châssis depuis assez longtemps, idem. Herbert Simon explique cette structuration avec le paradoxe de Tempus et Hora. Ce paradoxe conclut que les systèmes non chaotiques et stables s'organisent de manière hiérarchique pour résister aux interactions avec l'environnement.

D'autre part, les échanges entre un niveau système et sous-système sont souvent sans rapport. Par exemple, pensez-vous que les échanges des substances véhiculées par le sang dans votre corps partagent concrètement le moindre attribut avec les actions que vous réalisez tous les jours ? Quel rapport entre les attributs de la force appliquée sur une poignée de porte pour l'ouvrir et les attributs de la teneur en oxygène dans votre sang ? Aucun attribut, mais des liens fonctionnels bien sûr.

Tout cela correspond bien à la définition même du système : il est fait de sous-systèmes et déduire les propriétés du système à partir des interactions de ses sous-systèmes entre eux et avec l'extérieur n'est pas trivial.
Conclusion : chaque niveau système a ses propres attributs et ses propres interactions, et tout cela est assez stable dans le temps.

Cette structuration hiérarchique du monde qui nous entoure et qui est stable dans le temps est bien connue de tous. C'est même une clé de lecture pour nous pour le comprendre. Alors pourquoi cacher cette structuration dans le langage de description système. Au contraire, on gagne énormément à ce qu'elle soit explicite, que ce soit pour concevoir ou pour comprendre.

Les hiérarchies systèmes nous entourent et nous les manipulons partout : les "breakdown Structures" ou arborescences se retrouvent ainsi dans toutes les structurations d'artefacts de l'ingénierie système et projet (RBS, FBS, SBS, PBS, WBS, OBS…)

Selon nous cette notion de hiérarchie doit être citoyen de première classe dans le langage de description système. Non seulement cela simplifie le langage de description système et sa compréhension, mais en plus cela permet de créer des points de vue pertinents à peu de frais, cf. le point suivant.

Des points de vue cohérents plutôt qu'une seule hiérarchie complexe

Une fois que l'on dispose des hiérarchies systèmes comme "citoyen de première classe", on bénéficie du pattern-matching, de la récursivité et des langages fonctionnels sur ces hiérarchies. Ces notions sont mieux connues avec l'essor de l'IA ces dernières années car les langages fonctionnels et le pattern-matching sont à la base du machine learning.
Il y a d'ailleurs peut-être un lien entre les hiérarchies d'un réseau de neurone et la structure hiérarchique des systèmes qu'il observe et classifie ! La théorie des ondelettes dans la reconnaissance d'image permet d'identifier des contours d'objets de premier niveau, puis de second niveau… n'est-ce pas s'appuyer sur la structuration hiérarchique du monde qui nous entoure ?

La première application d'une hiérarchie explicite, ce sont les points de vue génératifs et cohérents que l'on peut créer instantanément à partir de n'importe quel modèle par pattern matching sur la structure. Par exemple, on peut visualiser sur la même base toutes les exigences allouées sur une hiérarchie système d'une part ou les interfaces de cette même hiérarchie d'autre part, ou la combinaison des deux. On peut encore visualiser tous les flux électriques, mécaniques, informatiques… d'une architecture séparément.
Les points de vue sont très importants pour abaisser la complexité de la modélisation et la rendre accessible aux utilisateurs.

Des chaines déduites de l'architecture et pas une multitude de diagrammes

Une fois un modèle établi, il est en général intéressant de créer des points de vue partiels que ce soit par rapport à la hiérarchie des objets ou par rapport aux interactions.
Le filtrage d'une partie de la hiérarchie correspond à un périmètre système qui concerne certains acteurs (par exemple, un responsable d'un sous-produit).
Le filtrage sur une partie des flux permet de se spécialiser sur un métier selon les types de flux. Un responsable des exigences se concentrera sur les liens de raffinement entre exigences. Un ingénieur électrique s'intéressera aux composants, flux, exigences et interfaces électriques…

Ces chaines sont à la fois une clé de lecture et un moyen d'éditer des informations sur un périmètre que l'utilisateur peut appréhender complètement.

Un langage formel bien-fondé et pas de concepts complexes sans traduction logique

Les modèles systèmes peuvent être très complexes. Il n'est pas rare qu'ils contiennent des centaines ou des milliers d'exigences, des dizaines ou des centaines de fonctions et systèmes, des centaines ou des milliers de flux, des dizaines ou des centaines de composants physiques et interfaces… Et surtout, un ordre grandeur en plus du nombre d'objets pour caractériser toutes les relations : allocations, productions de flux…

Nous sommes donc en face d'une structure d'information qui peut contenir couramment des dizaines de milliers d'objets et de liens si ce n'est des centaines de milliers ou des millions.

Comment s'assurer de sa cohérence et de sa conformité à un ensemble de règles ? Par exemple toute exigence doit être allouée… mais bien sûr, cela peut être bien plus complexe. Par exemple un dossier de sécurité devra couvrir la décomposition des objectifs de sécurité en exigences allouées sur chaque composant et chaque interface.

Dans ces conditions, le langage qui héberge la description de votre système doit être suffisamment simple et bien-fondé pour permettre une intégration dans un système formel et la preuve des propriétés que vous souhaitez démontrer pour votre modélisation système.

arKItect est la seule plateforme qui implémente ces principes. Elle est éprouvée avec plus de 16 ans d'existence, des millions d'heures d'utilisation et des clients opérationnels et fidèles depuis plus de 10 ans.

Elle dispose de manière intégrée d'autres atouts importants :

- Une gestion des options et variantes
- Une gestion de configuration
- Une API python permettant de générer des documents ou d'exécuter des programmes simplement
- Une interface ouverte vers les autres outils et notamment des fonctions d'import/export Excel configurables simplement.
- Une utilisation collaborative avec gestion des collisions lorsque plusieurs personnes veulent modifier le même objet ou la même vue