Systems Architecting

Abdellatif MEGNOUNIF

Chap. 7

Systèmes de technologie de l'information et du Software

COURS 7 **Jeudi 05.01. 2012**

1. Introduction

- Software est devenu la partie importante de systèmes complexes, dans le sens que n'importe quelle évolution de performance ne peut se faire sans le software.
- On s'intéresse de plus en plus en développement de software que de hardware. Ceci est du à la capabilité du software de créer un comportement intelligent et d'adapter rapidement les tendances technico économiques dans le développement de hardware.
- Actuellement, il ya 70% de software et 30% de hardware (alors que c'était l'inverse).
- Il faut juste voir comment l'industrie des semi conducteurs a énormément changée. On utilise des microprocesseurs hautement intégrés avec plus de périphériques sur la puce.
- Les produits basés sur les microprocesseurs obtiennent leurs fonctionnalité par le software qu'ils exécutent.

1. Introduction (suite).

- Ainsi le développeur de produit est passé de designer de hardware à l'intégrateur de hardware et développeur de software.
- Comme la bibliothèque de développement de software devient de plus en plus importante, plus capable et acceptée, beaucoup de développeurs de software seront convertis en intégrateurs de software.
- Le plus grand marché de software actuel est appelé « technologie de l'information ».
- Il regroupe le domaine des computers et des communications appliqué aux entreprises publiques commerciales.
- L'architecture du software est lié au développement de software original que construire des systèmes de traitement de l'information à travers l'intégration de composants de software et hardware importants.
- En pratique, la technologie de l'information est moins concernée par le développement d'applications originales complètes mais par la construction de systèmes par intégration.

1. Introduction (suite).

- **Le software possède 02 attributs clés:**
  1. **Un software bien architecturé peut rapidement évoluer du à sa facilité de remplacement (annuel ou trimestriel) (mises à jour).**
  2. **Software est un milieu flexible exceptionnel. Il peut avoir plusieurs concepts complexes logiques comme les langages à couches, exécution de la règle conduite, relations entre données et autres…cette flexibilité d'expression fait du software un excellent milieu pour implémenter l'intelligence des systèmes.**
- **Aussi la combinaison des tendances techniques et économiques favorise la construction de systèmes à partir de hardware standards et un software de système unique.**
- **Une conséquence de la complexité croissante du software et son rôle central, est la reconnaissance de l'importance de l'architecture du software et son rôle dans le design de systèmes.**

1. Introduction (suite).

- Architectes, architectures et architecting sont des termes qui deviennent de plus en plus utilisés dans le domaine du software.
- Architecture du software a été bien définie et de façon formelle.
- L'architecture est la structure générale d'un système de software en termes de composants et interfaces. Ça inclut les composants du software majeurs, leurs interfaces entre elles et le monde extérieur et la logique de leur exécution.
- Il ya plusieurs exemples de systèmes centrés sur le software (militaires ou civiles) surveillance globale, communications de défense, les super autoroutes de l'information, Internet, téléphonie cellulaire globale, opérations de fabrication flexibles…
- Les conséquences de cette évolution rapide sont les systèmes intelligents.
- Le software n'a plus un rôle de support uniquement (généralement après avoir fixé le hardware) mais plutôt devient la pièce centrale du design et opération de systèmes complexes.

1. Introduction (suite).

- Bien qu'on s'intéresse de plus en plus au software, le client finalement ne cherche que le système lui-même (produit final).
- Les 02 (système et software) partagent beaucoup de racines communes mais leurs demandes différentes les a conduit vers des directions différentes.
- Une partie du rôle de l'architecting des systèmes est de les ramener ensemble de manière intégrée.
- L'ingénierie du software est une source riche pour les modèles intégrés; modèles qui combinent, lient et intègrent plusieurs visions du système.
- Plusieurs formalismes utilisés en ingénierie de software sont utilisés en ingénierie des systèmes.
- Dans ce chapitre on va voir les différences entre l'architecting de systèmes et l'architecting de software, les orientations actuelles dans l'architecting de software et les heuristiques et les lignes directrices pour le software.

2. Le software comme composant du système

- **Comment l'architecting de software interagit avec l'architecting du système global ? Le software a des propriétés uniques qui influencent la structure du système global.**
  1. **Le software offre une palette d'abstractions pour la création du comportement de système. Le software est extensible à travers une programmation par couches pour fournir des environnements de développement et d'interfaces d'utilisateur abstracts.**
  2. **C'est économiquement et techniquement faisables d'utiliser la livraison évolutive du software. Un composant de software d'un système déployé peut être complètement remplacé en un horaire régulier.**
  3. **Software ne peut pas fonctionner indépendamment. Il doit toujours être résident dans un système hardware. (donc intégré avec). Les interactions entre eux deviennent les éléments clés de design de systèmes basés sur le software.**

2. Le software comme composant du système. (suite)

2.1 Software pour les systèmes modernes

- Le software joue des rôles disparates dans les systèmes modernes.
- Software d'application du marché de masse, software de contrôle et d'analyse en temps réel, assistants interactifs aux humains sont tous des systèmes basés sur le software mais chacun est différent de l'autre.
- les attributs du software de la fonctionnalité riche et la susceptibilité d'évolution correspondent bien aux caractéristiques des systèmes modernes qui sont:
  1. Stockage de, et semi autonomie et interprétation intelligente de, volumes larges d'information.
  2. Fourniture d'interfaces humaines réactives qui masquent les machines sous jacentes et présentent leur fonctionnement dans la métaphore.

2. Le software comme composant du système. (suite)

2.1 Software pour les systèmes modernes (suite)

3. **Adaptation semi autonome au comportement de l'environnement et des utilisateurs individuels.**
4. **Contrôle en temps réel du hardware à des taux au-delà de la capabilité humaine avec une fonctionnalité complexe.**
5. **Construit de composants informatiques à grande production et software de système unique avec la capabilité d'être personnalisé pour des clients individuels.**
6. **Coévolution des systèmes avec les clients comme expérience avec les perceptions des changements de technologie du système de ce qui est possible.**

2.2 Modèles du procès, du software et des systèmes.

- Le défi architectural est de réconcilier les besoins d'intégration du software et du hardware pour produire un système intégré.
- Ceci est un problème de représentation et modélisation et en même temps de procès.
- Coté procès, hardware est mieux développé avec peu d'itérations alors que le software peut évoluer par plusieurs itérations.
- Hardware doit suivre un cycle de production et de design bien planifié pour minimiser les couts, avec une production à grande échelle reportée le plus prés de la livraison finale.
- Mais le software ne peut pas être développé fiablement sans accéder à la plateforme du hardware pour beaucoup de son cycle de développement.
- Les couts de distribution du software sont comparativement moins élevés de telle façon que les mises à jour sont devenues normales.
- Alors que les couts du hardware sont souvent dominés par la production physique du hardware. (et non pas par la partie développement). Il est donc déconseiller de remplacer un hardware déployé avec des modifications mineurs.

Modèles en cascades pour le software ?

- Pour les systèmes hardware, le modèle du procès est en cascades qui essaye de garder les itérations locales: i.e. entre les taches adjacentes comme les exigences et design.
- Une fois atteindre la production, il n'ya pas de notion d'itérations, sauf celle à grande échelle de l'évaluation du système et de son retrait et/ou remplacement.
- Software peut avoir un modèle en cascade de développement. On a souvent adopté le paradigme séquentiel des exigences, design, codification, test et livraison.
- Mais l'insatisfaction avec le modèle en cascade pour le software a conduit au modèle en spirale et ses variantes.
- Plus spécialement, tous les systèmes software réussis sont livrés itérativement.

Modèles en cascades pour le software ?

- Une raison des itérations du software est de fixer les problèmes rencontrés sur le champ.
- Le modèle en cascade essaye d'éliminer ces pbs en faisant un travail de très grande qualité des exigences.
- Le succès du développement en cascade dépend fortement de la qualité des exigences.
- Mais dans certains systèmes, l'évolubilité du software peut être exploité pour atteindre rapidement le marché et éviter des recherches en exigences couteuses et peut être non fructueuses.

Modèles en spiral pour le hardware ?

- L'utilisation du modèle en spiral pour l'acquisition du hardware est équivalente au prototypage répété.
- Le système à hardware intensif (un du genre) ne peut être prototypé de la manière usuelle. C'est un système où, du premier article, chacun doit être produit comme si c'est le seul article.
- Un prototype pour ce type de système doit être une version limitée ou bien un composant destiné à répondre à des questions de développement spécifiques.
- Le procès de développement doit mettre l(accent sur le développement des exigences et une attention particulière sur les objectifs détaillés à travers le cycle de design.
- Les systèmes à production de masse ont de grandes latitudes en prototypage à cause du rapport de cout du prototype/production.

Modèles en spiral pour le hardware ?

- L'utilisation du modèle en spiral pour l'acquisition du hardware est équivalente au prototypage répété.
- Le système à hardware intensif (un du genre) ne peut être prototypé de la manière usuelle. C'est un système où, du premier article, chacun doit être produit comme si c'est le seul article.
- Un prototype pour ce type de système doit être une version limitée ou bien un composant destiné à répondre à des questions de développement spécifiques.
- Le procès de développement doit mettre l(accent sur le développement des exigences et une attention particulière sur les objectifs détaillés à travers le cycle de design.
- Les systèmes à production de masse ont de grandes latitudes en prototypage à cause du rapport de cout du prototype/production mais moins que le software.

2. Le software comme composant du système. (suite)

2.2 Modèles du procès, du software et des systèmes. (suite)

Intégration : Spiraux et les cercles.

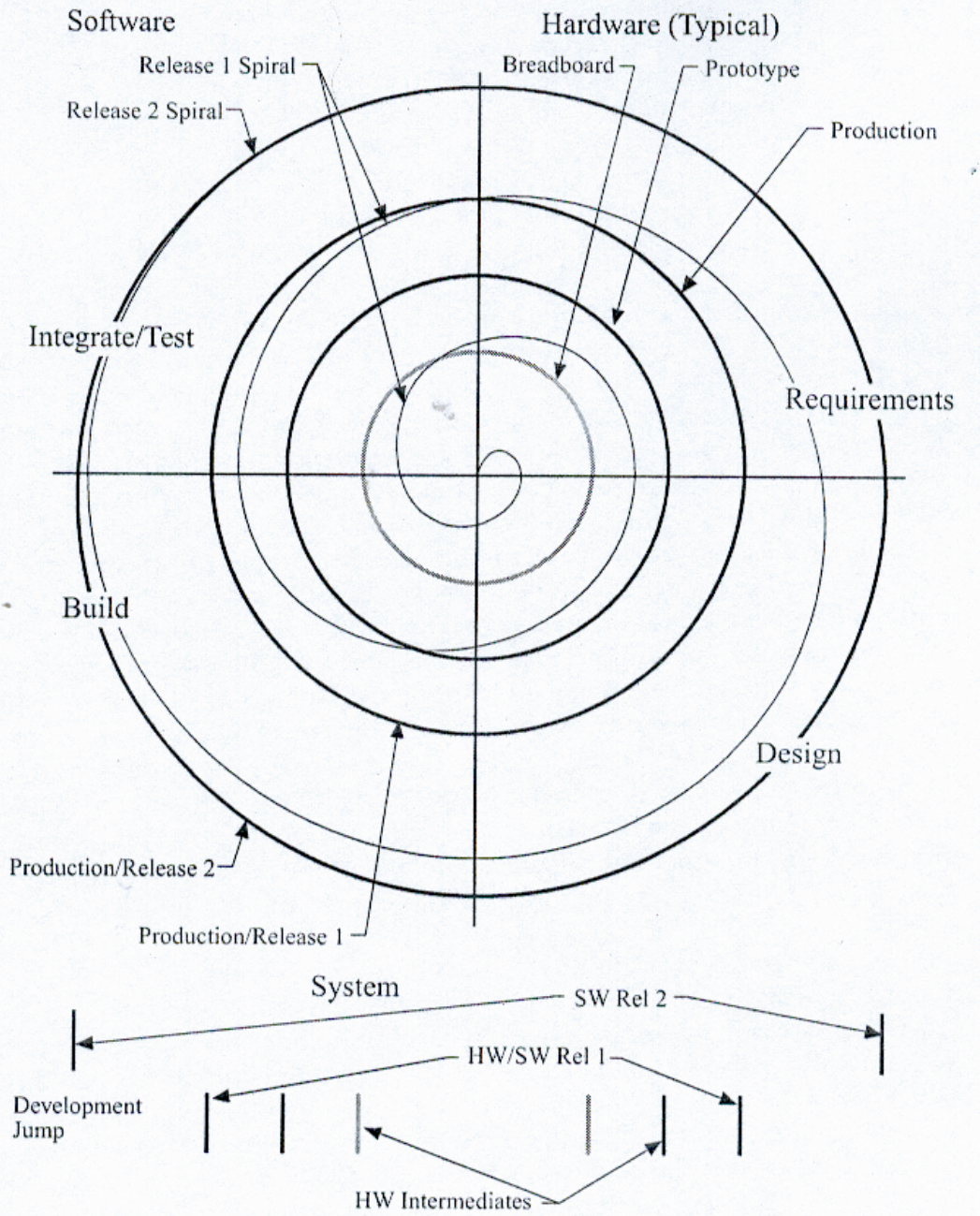
- Quel sera le modèle du procès qui correspond bien aux systèmes complexes en comportement, de technologie combiné et de nature évolutive ? Le modèle spiral au cercle.
- Le système doit avoir des configurations tables (représentées par des cercles) et le développement doit approcher ces cercles de façon itérative.
- Les configurations stables peuvent être des niveaux de version du software, des cadres architecturaux ou bien des configurations de hardware.
- Ce modèle de procès correspond bien à la pratique du software admise, celle de se déplacer à travers des niveaux de versions définis où chaque version est produite en cycles de exigences-design-code-test.
- Chaque niveau de version est une forme stable qui sera utilisée pendant le développement de la prochaine version.

2. Le software comme composant du système. (suite)

2.2 Modèles du procès, du software et des systèmes. (suite)

Intégration : Spiraux et les cercles.

- Un produit software (ex: operating system) a des incréments majeurs en comportement (indiqué par le changement du numéro de la version) et plus d'incréments mineurs en changeant le numéro après le point. (ex: produit de version 7.2: version 7 majeure et 2ème mise à jour).
- Les version majeurs peuvent représentés des cercles où les versions mineurs tournent à l'intérieur d'eux.
- Au 3ème niveau, il ya les changements qui conduisent à de nouveaux systèmes ou à des anciens systèmes réachitecturés.
- La figure montre que les versions majeurs représentent des sauts verticaux. Et la spirale évolutive se déplace aux configurations majeurs stables et puis saute au prochain changement majeur.
- En pratique, l'évolution d'un niveau de version peut avoir simultanément avec le développement d'un changement majeur.



2. Le software comme composant du système. (suite)

2.2 Modèles du procès, du software et des systèmes. (suite)

Intégration : Spiraux et les cercles.

- Il ya aussi le problème de l'intégration hardware-software.
- Les configurations du hardware doivent aussi être stables mais doivent apparaitre à des points différents de temps que les intermédiaires du software.
- Certains hardware stables doivent être disponibles pendant le développement du software pour faciliter ce développement.
- Une des solutions à l'intégration et d'architecturer ensemble le procès et le produit.
- Le procès est manipulé pour permettre différents segments du développement pour se correspondre aux demandes de la technologie d'implémentation.
- Le produit est conçu avec les interfaces qui permettent la séparation des efforts de développement lorsque les efforts doivent procéder sur des voies différentes.

2. Le software comme composant du système. (suite)

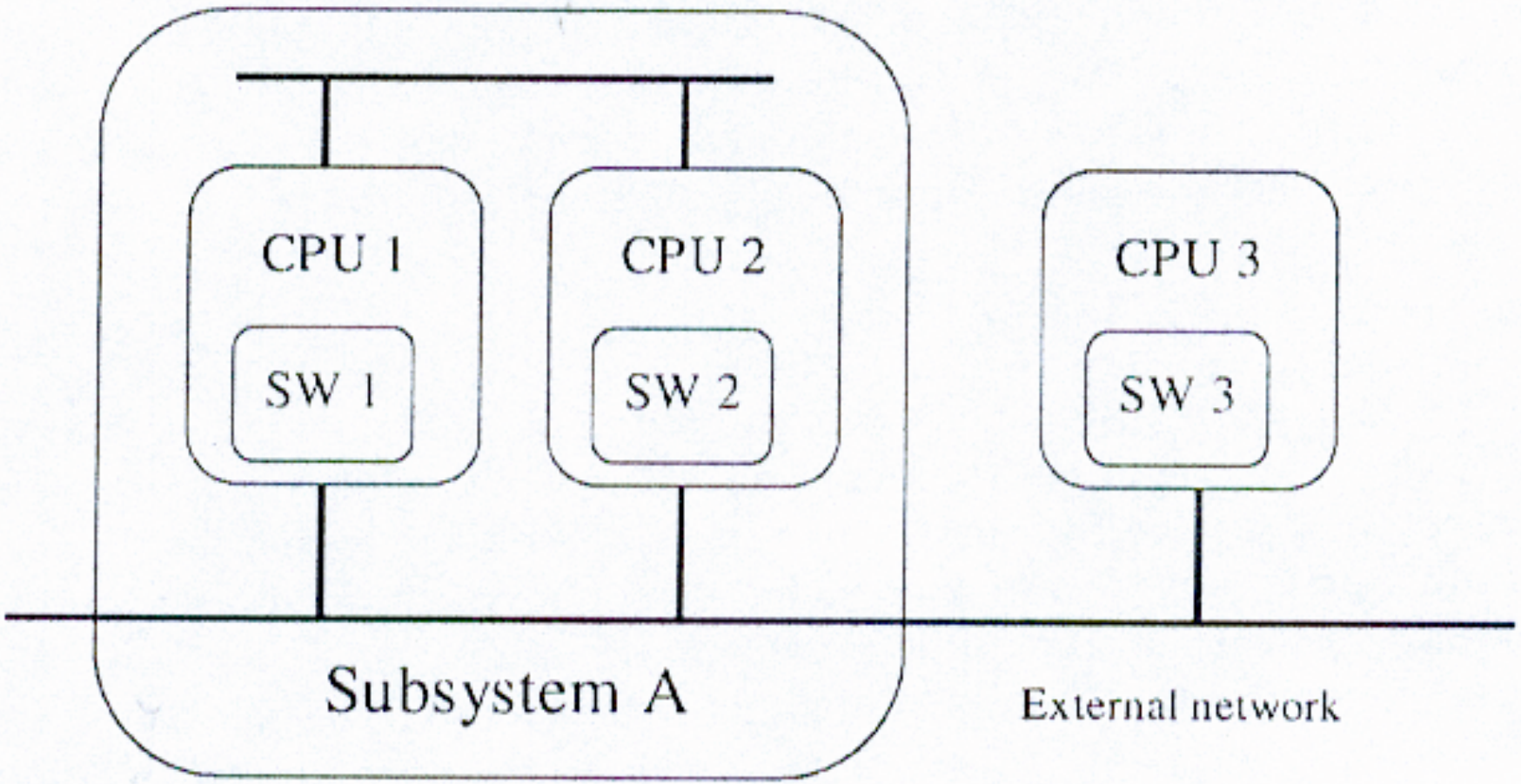
2.3 Le problème de la hiérarchie.

- Un principe central en ingénierie des systèmes est que tous les systèmes peuvent être vus en hiérarchies. Un système a des sous systèmes eux-mêmes ont des sous sous systèmes…
- Une stratégie de base est de décomposer le système en sous systèmes…puis intégrer par inverse. Les composants sont intégrées dans le niveau supérieur de sous systèmes, eux-mêmes intégrés dans les niveaux supérieurs jusqu'à atteindre l'assemblage total du système.
- La logique de décomposition-intégration est aussi utilisée dans le développement de certains software. Exemple le cas des langages « C » et « Pascal ».
- Dans ces systèmes software le code commence avec des routines top-level qui appellent le 1er niveau de routines, eux appellent le 2ème niveau de routines jusqu'au dernier niveau de routine qui n'appellent pas d'autres.

2. Le software comme composant du système. (suite)

2.3 Le problème de la hiérarchie. (suite)

- Dans ce cas les routines du niveau le plus bas sont contenues dans les routines de niveau supérieur.
- En écrivant le code, la chaine de décomposition se termine en composants comme la décomposition du hardware qui se termine en composants.
- De même, on peut intégrer et tester le système de software de la même façon du niveau bas et plus élevé (bottom-up).
- on voyait de façon classique la position du système software dans le système hardware (figure).
- Les unités du software sont contenues dans les unités du processeur qui les exécutent.
- Le software est donc considéré comme un sous système du processeur.
- Actuellement, c'est très différent avec les software modernes. Ceux écrits en orienté objet, construits en couches…le problème est différent et ne ressemble nullement à une décomposition hiérarchique



2. Le software comme composant du système. (suite)

2.3 Le problème de la hiérarchie. (suite)

Orientation objet.

- Dans le sens de l'architecting des systèmes, un objet est une collection de fonctions (appelées méthodes) et des données. Certaines de ces fonctions sont publiques
- Les objets peuvent être actifs, i.e: ils peuvent être exécutés de façon concurrente avec d'autres objets.
- L'exécution concurrente des objets est passée au système d'exploitation distribué qui peut contrôler l'exécution de l'objet à travers des politiques définies séparément.
- En systèmes orienté objet, le nbr d'objets existants lors de l'exécution de software est indéterminé.
- Un objet a des classes, un modèle défini. C'est l'équivalent du « type » dans une programmation procédurale.
- Comme la déclaration de plusieurs types, il ya la déclaration de plusieurs objets correspondants à une classe.

Orientation objet.

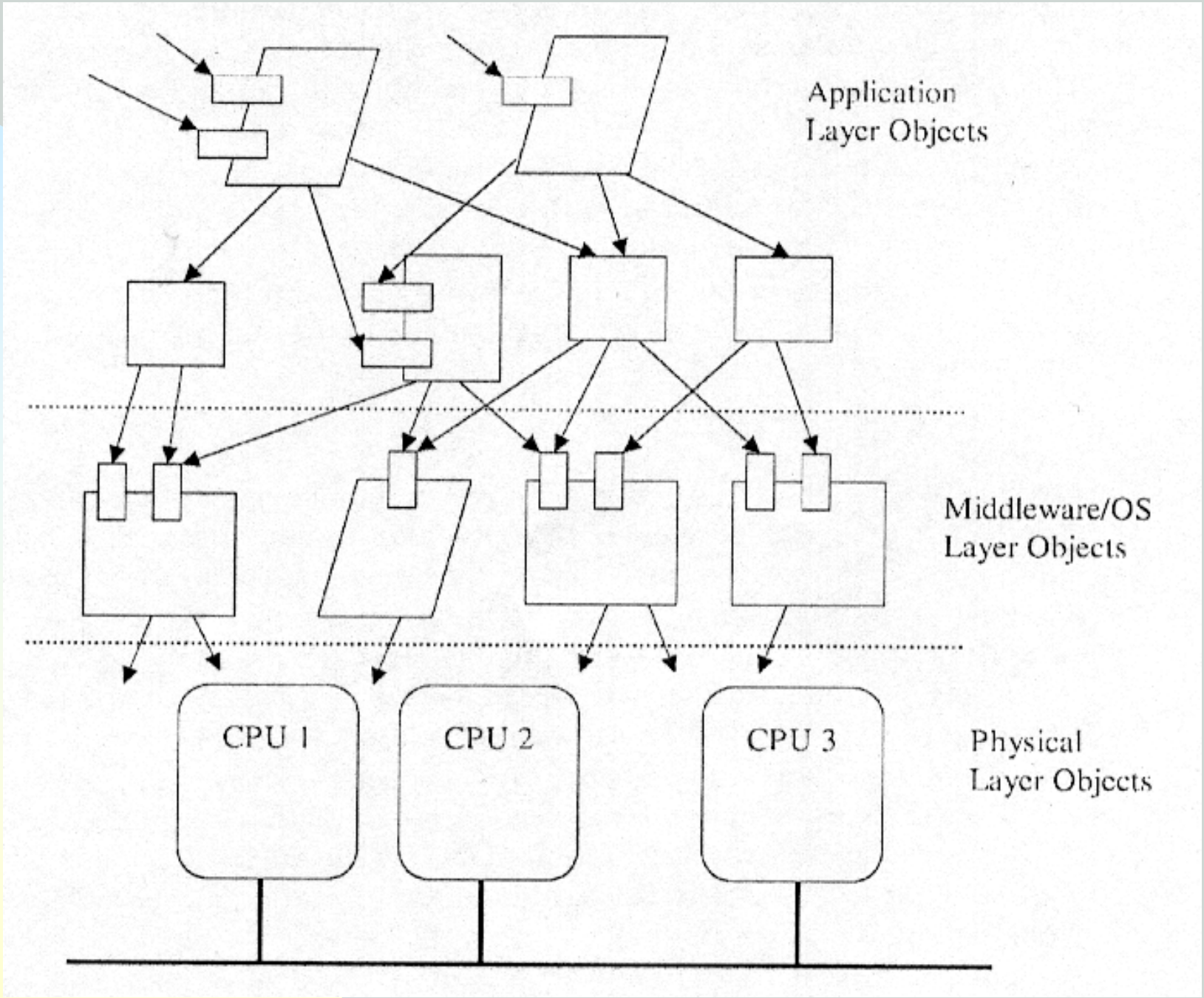
- Les langages orientés objet, la création d'objets des classes se produit pendant le temps d'exécution qd le software s'exécute.
- Si les objets ne sont créés qu'au temps d'exécution, leur nbr peut être contrôlé par des événements extérieurs.
- c'est une méthode très puissante pour la composition d'un système software. Chaque objet est une machine à calcul. Il a ses propres données et un code de programme.

Design en couches

- Ces composants de services middleware s'exécutent sur les couches inférieures du software du réseau, fournis comme une partie des services du système d'exploitation.
- En général, le pb d'hiérarchie est que la hiérarchie du software et celle du hardware à un point vont être déconnectées.
- Pour l'architecte du software, la structure naturelle du système est des couches d'objets concurrents (figure).
- i.e: les architectes de systèmes et de software peuvent entrer en conflit en partitionnant le système et des contraintes inappropriées peuvent être placées sur l'un ou l'autre.

Design en couches

- Les objets sont composés en design en couches.
- Les couches sont une forme d'hiérarchie avec une différence critique.
- En système en couches, les éléments du niveau le plus bas (ceux formant la couche la plus basse) ne sont pas contenues dans les éléments de la couche d'avant.
- En orienté objet, par contre il ya la notion d'encapsulation. Un objet a des internes et les internes (fonctions et données) appartiennent uniquement à cet objet bien qu'ils peuvent être dupliqués dans d'autres objets de la même classe.
- Une application distribuée moderne peut être construite comme un groupe d'objets concurrents en interaction.
- Les objets eux-mêmes peuvent interagir avec la couche plus bas, appelée « services middleware » qui sont fournit par des unités de software externes.



2. Le software comme composant du système. (suite)

2.3 Le problème de la hiérarchie. (suite)

Composants autonomes, grands.

- Dans l'approche « décomposition », le concepteur vers le niveau le plus bas pour atteindre les éléments du système qui sont facilement achetés ou construits.
- En software ou hardware, certains composants sont grands. Des unités importantes comme systèmes d'exploitation et bases de données.
- Ils peuvent agir en semi autonomie qd ils sont utilisés dans un système.
- Ce qui influe sur l'architecture. Les architectes sont forcés d'adapter à ces composants.
- La dominance du marché et la complexité des grandes bases de données nous forcent d'utiliser les produits commercialisés dans ces applications.
- L'architecture doit prendre en considération les types de modéls de données supportés.

Concilier les hiérarchies

- Le défi est de concilier le software et le système.
- Parce que le software devient l'élément dominant, on pourrait se pencher d'adopter le modèle du software et abandonner la vision classique du système.
- Ceci est inapproprié et pour plusieurs raisons.
  1. Le passage du software aux structures en couches et orientées objet n'est que partiel.
  2. Les 02 approches sont fondamentalement valides. Il n'ya pas de vision favorisée.
  3. Chaque partitionnement a ses avantages et ses inconvénients

3. Le rôle de l'architecture dans les systèmes centrés sur le software.

- En software, comme en systèmes, le rôle de base de l'architecte est la réconciliation de la forme physique avec les besoins du client pour la fonction, le cout, la certification et la faisabilité technique.
- Certaines heuristiques du système restent valables pour le software d'autres sont du software propre.
- La responsabilité de l'architecte va au-delà de l'intégrité conceptuelle des systèmes comme vu par l'utilisateur mais à l'intégrité conceptuelle comme vu par le constructeur et les autres intervenants.
- L'architecte est responsable de ce que fait le système et de comment il le fait.
- La vision est une vision système où on cherche le software avec la plateforme hardware correspondante dans une vision intégrée.

3. Le rôle de l'architecture dans les systèmes centrés sur le software. (suite)

- Architecturer l'évolution est un exemple de l'heuristique « the greatest leverage (effet du levier) is at the interfaces ».
- Rend un système évolutif en faisant attention aux interfaces.
- En hardware les interfaces sont de type communication au niveau bite, byte et message.
- En software, les interfaces sont diverses et peuvent être plus riches et captent des données structurées extensivement .
- Exemple le « java virtuel machine ». Langages portable network (comme le java) permettent à chaque machine d'exprimer une interface commune pour le code du mobile.

3.1 Langages de programmation, modèles et expressions

- Modèles sont des langages.
- Un langage de programmation est un modèle de la machine à calculer.
- Le développement des langages de programmation se fait en allant plus haut dans l'abstraction du hardware.
- La superposition des langages est essentiel pour le développement de software complexes.
- L'assemblage des langages masquent les instructions de la machine alors que les langages procédurales modèlent les instructions du computer autrement.
- Matlab est un exemple.

3. Le rôle de l'architecture dans les systèmes centrés sur le software. (suite)

3.1 Langages de programmation, modèles et expressions (suite)

- Une des heuristiques les plus utilisées en programmation est : « Programmers deliver the same number of lines of code per day regardless of the language they are wrinting in ».
- Ainsi, pour atteindre une productivité du software élevé, les programmeurs doivent travailler avec des langages nécessitant moins de lignes du code.
- La nature du langage et les outils et librairies disponibles va déterminer la quantité de codes nécessaire pour une application donnée.
- Actuellement, il ya d'autres tendances. Ex: les tableurs qui combinent l'abstraction visuelle et le langage de programmation.
- Il ya aussi les langages mathématiques, MatLab, Mathematica, MacSyma…permettent à l'utilisateur de passer de la syntaxe mathématique à une forme de langage computer.

3.2 Architectures, modèles unifiant et visions

- Les architectures en software peuvent etre des définitions en termes de taches et modules, langages ou modèle de construction ou bien au plus haut niveau d'abstraction, des métaphores.
- Un exemple est le métaphore Macintosh desktop.
- La puissance du métaphore comme architecture est double:
- Le métaphore suggère plus qu'il va suivre. Si le métaphore est un desktop, ces composants doivent fonctionner de façon similaire que ces homologues physiques familiers.
- Il fournit un modèle communicable facile du système où chacun peut utiliser pour évaluer l'intégrité du système.

4. Directions en architecting de software

- La plus part des travaux actuels en architecture de software s'intéressent aux structures architecturales et leurs analyses.
- Le travail en styles architecturaux de software est d'essayer de trouver et de classer les formes de niveau élevé du software et leurs applications à des pbs particuliers de software.
- Le travail en programmation structurée conduit au design structuré et aux modèles multitâches et orientés objet.
- La tendance actuelle de l'architecture de software est d'adresser le produit de l'architecting (la structure de l'architecture) au lieu du procès de sa génération.
- On s'intéresse plutôt à la classification des styles architecturaux, aux chemins et langages de chemin en software et les structures du software.

4.1 Styles architecturaux

- À un niveau général, le style est défini par ses composants, connecteurs et contraintes.
- Les composants sont ceux avec lesquels le systéme software est composé.
- Les connecteurs sont les interfaces avec lesquelles les composants interagissent.
- Le style définit les types de composants et connecteurs qui formeront le système.
- Les contraintes sont les exigences qui définissent le comportement du système. En utilisation courante, l'architecture est la définition en termes de forme qui n'inclut pas explicitement les contraintes.

4. Directions en architecting de software. (suite)

4.1 Styles architecturaux (suite)

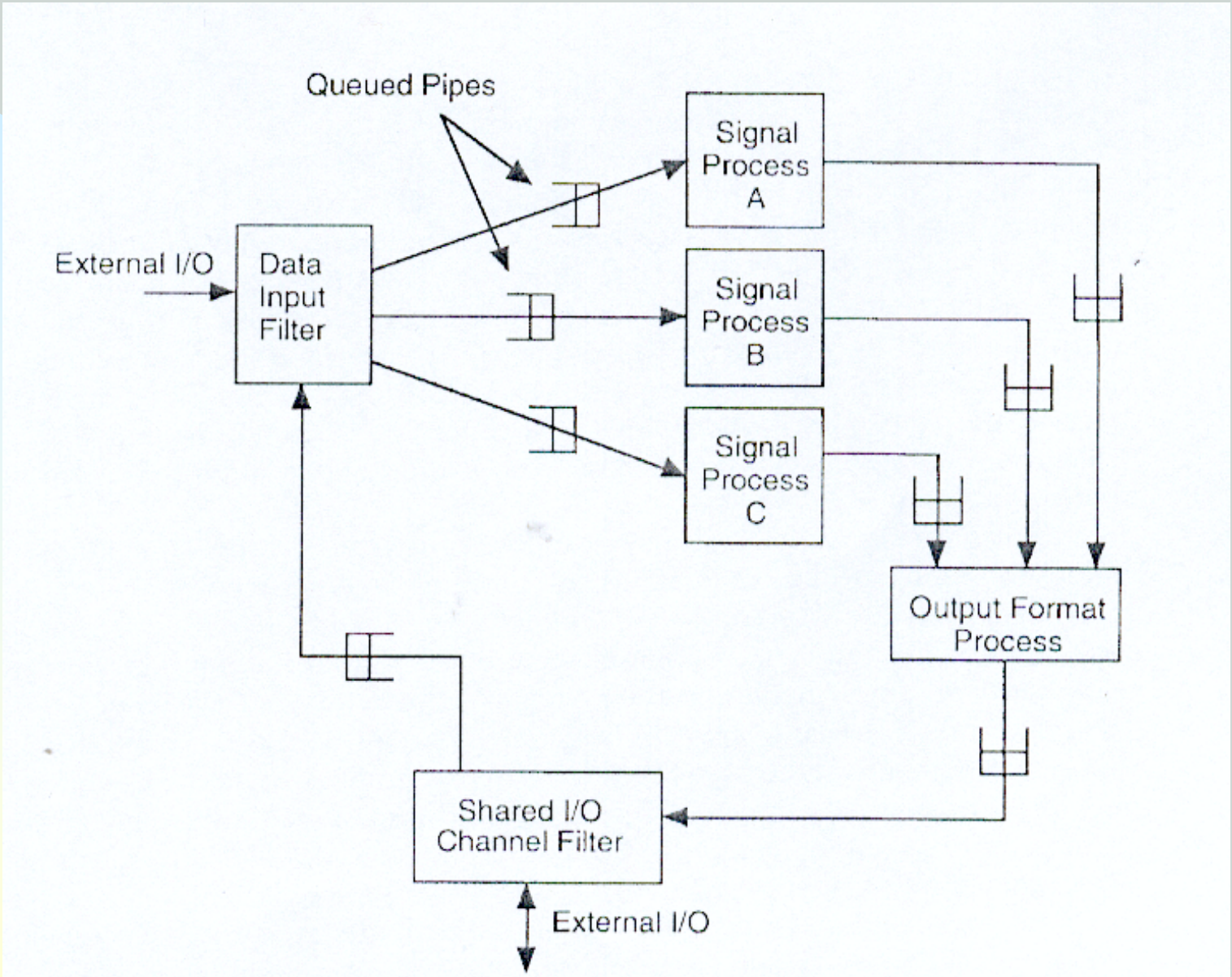
- **Par définition (David Garlan et Mary Shaw) :**
  - **Un style architectural définit une famille de ces types de systèmes en termes de modèle d'organisation structurelle.**
  - **Spécifiquement, un style architectural détermine le vocabulaire des composants et connecteurs qui peuvent être utilisés dans les instances de ce style.**
  - **Additionnellement, un style peut définir des contraintes topologiques sur les descriptions architecturales.**
- **Avec ceci, on peut comprendre ce que c'est un style en répondant à ces questions suivantes:**

  Quel est le modèle structurel – les composants, les connecteurs et les topologies? Quel est le modèle sous jacent de calcul? Quels sont les invariants essentiels du style? Quels sont les exemples courants de son utilisation? Quels sont les avantages et inconvénients de son utilisation?

4. Directions en architecting de software. (suite)

4.1 Styles architecturaux (suite)

- Un exemple de style est celui du « pipe and filter ». Il contient un seul type de composant (filtre) et un seul type de connecteur (le tuyau).
- Chaque composant input et output des flux de données.
- Tous les filtres peuvent fonctionner de façon incrémentale et concurrentielle.
- Les flux s'écoulent à travers les tuyaux. Les courants de flux sont aussi potentiellement concurrents.
- Parce que chaque composant agit pour produire un ou plusieurs flux à partir d'un ou plusieurs flux, on peut penser à une sorte d'abstraction du filtre. (figure)
- Un exemple de systèmes « tuyau et filtre » est le système UNIX.
- D'autres styles sont « les orientés objet », « event-based », « layered » et « blackboard »…



4. Directions en architecting de software. (suite)

4.1 Styles architecturaux (suite)

- Chaque style a ses avantages et ses faiblesses.
- Pour réussir, il ne faut pas considérer que les exigences précédent l'architecture. Le développement des exigences fait partie de l'architecture et non pas partie de ses pré conditions.
- le style idéal est celui qui combine les visions de l'utilisateur et du constructeur. Les langages mathématiques (MatLab…) sont de meilleurs exemples.

5. Heuristiques en software

- **La littérature software est une source très riche pour les heuristiques.**
  - **Choose components so that each can be implemented independently of the internal implementation of all others.**
  - **Programmer productivity in lines of code per day is largely independent of language. For high productivity, use languages as close to the application domain as possible.**
  - **The number of defects remaining undiscovered after a test is proportional to the number of defects found in the test, but rarely less than 0.5.**
  - **Very low rates of delivered defects can be achieved only by very low rates of defect insertion *throughout* software development, and by layered defect discovery – reviews, unit test, system test.**
  - **Software should be grown or evolved, not built.**

5. Heuristiques en software (suite)

- The cost of removing a defect from a software system rises exponentially with the number of development phases since the defect was inserted.
- The cost of discovering a defect does not rise. It may be cheaper to discover a requirements defect in customertesting than in any other way, hence the importance of prototyping.
- Personnel skill dominates all other factors in productivity and quality.
- Don’t fix bugs later, fix them now.

- A system will develop and evolve much more rapidly if there are stable intermediate forms than if there are not.

Systems Architecting

Abdellatif MEGNOUNIF

Semaine Prochaine

Systèmes collaboratifs.

COURS 8 **Jeudi 12.01.2012**

Merci. Fin du chapitre 7