IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

(Software) Architecture

Fabien Dagnat

ILSD – FIAB – 3– 2025-2026

Avancement

Qu’est-ce que l’architecture ?

- C’est
  - une phase initiale dans le cycle de vie d’un projet

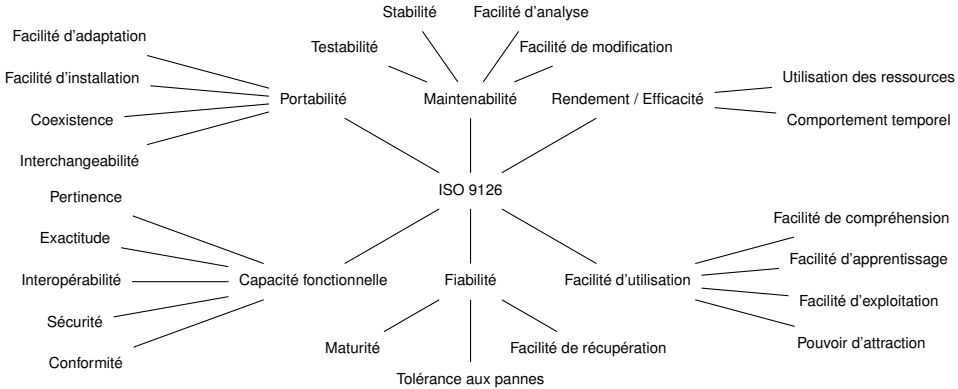
    Analyse $\longrightarrow$ Architecture $\longrightarrow$ Conception $\longrightarrow$ $\cdots$

  - le résultat de cette phase
- Le but est de prendre du recul, voir le futur système de loin
- Les décisions d’architecture sont des compromis coûteux à remettre en cause
- On se concentre sur le pourquoi par rapport au comment (plutôt conception)
- Une architecture forme un ensemble de contraintes pour les phases suivantes
- Attention, architecture $\neq$ conception : vision stratégique versus tactique

Les dimensions de l'architecture

1. Quelles sont les caractéristiques voulues de notre système ?
2. Quels sont les éléments logiques du système ?
3. Quel est le style de l'architecture de notre système ?
4. Quelles sont les décisions (contraintes) prises ?

Propriétés non-fonctionnelles ISO 9126



- Souvent en conflit $\Rightarrow$ il faut leur donner des niveaux de priorité
- Il faut les mesurer, les tester, les surveiller ...

Les éléments logiques d'une architecture

- Permettent de décrire le comportement du système (sa *fonctionnalité*)
- Quels éléments ?
  - systèmes, composants, fonctions
  - interactions avec les utilisateurs, l'environnement ou entre éléments
  - emplacements (conteneurs, machines virtuelles, serveurs, fournisseurs cloud ...)
  - connexions (réseaux, protocoles, VPN, débit ...)
- Attention, logique $\neq$ physique

La notion de style

- Ensemble de règles à respecter sur
  - la topologie des composants
  - l'architecture physique
  - le déploiement
  - la / les formes de communication
  - la topologie des données
- Un style apporte des caractéristiques
- Il existe des catalogues

Les décisions architecturales

- Ensemble de contraintes sur la réalisation du système

L’architecte

- Prends des décisions (après étude)
- Maintient les descriptions et justifications
- Surveille la réalisation du système et vérifie le respect des contraintes
- Connais largement la technologie, l’environnement, le domaine du système
- Gère des équipes, des projets ...

Avancement

Interaction et interface

- Deux éléments, actions ou influence réciproque (échange)
- Comment on interagit
- Les systèmes (ou composants) interagissent via des interfaces
- Une interface définit les règles, les hypothèses et le contexte des interactions
- En architecture, on cherche à décrire les interfaces

Interface

- De multiples formes, interface envers des utilisateurs, interface électrique, mécanique, logiciel !
- Crucial de s'y interesser car source de complexité, on ne maîtrise pas les autres parties
- Doivent être définie avant car fournissent les hypothèses pour ceux qui font les éléments

Des soucis fameux

- Retard de l'A380 car assemblage de pièces impossibles
  - Complexité du câblage : 530km, 100000 cables et 40300 connecteurs
  - Usage de deux versions différentes du logiciel Catia (erreur d'alignement)
  - `https://www.bloomberg.com/news/articles/2006-10-04/airbus-first-blame-the-software`
- Satellite US/EU
- Couplage de 2 rames de TGV

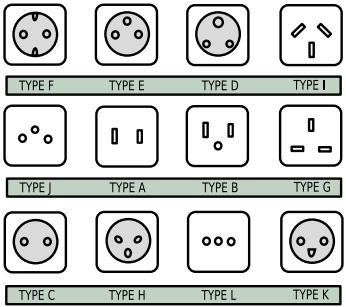
But d'une interface

- Assurer le couplage
- Assurer l'interopérabilité
- Mettre les 2 parties d'accord
  - Type de donnée, unité, ordre des messages (protocole), intensité, etc.
- Plusieurs points de vues
  - Physique/chimie
  - Science de la communication
  - Électronique
  - Informatique

Spécification

- Description de tout ce qui est nécessaire au bon usage de l'élément
  - Mécanique (poids, résistance mécanique, dimensions, nature des matériaux, frottement ...)
  - Électrique (tension, intensité, sens, évolution du signal ...)
  - Information (codage, lexique, sens, ordre d'utilisation, qualité de service ...)
- Les questions : *quoi*, *comment*, *où*, *quand*, *qui* doivent trouver une réponse
- Et la question *pourquoi* ?
- But : décrire pour vérifier la comptabilité des connexions

Comptabilité d'assemblage



- Bibliothèque de code C `double sqrt(double)`
- Qu'est-ce que cela garantit ?
- La correction de l'édition de lien

En Informatique, le contrat d'interface

1. Physique (en général abstrait, caché)
2. Logique (noms) ⇒ niveau syntaxique : `double sqrt(double)`
3. Sémantique (sens) ⇒ niveau sémantique : pré, post conditions
   `pre: x >= 0  post: result >= 0 && x = result * result`
4. Synchronisation : règles d'utilisation (concurrence, protocole ...)
5. Qualité de service : propriétés quantitative (efficacité, fiabilité, disponibilité ...)

- Du côté du producteur, ce qu'il fournit
- Du côté du client, ce qu'il exige
- Description en langue naturelle, sous forme d'API, en modèle (UML, SysML)

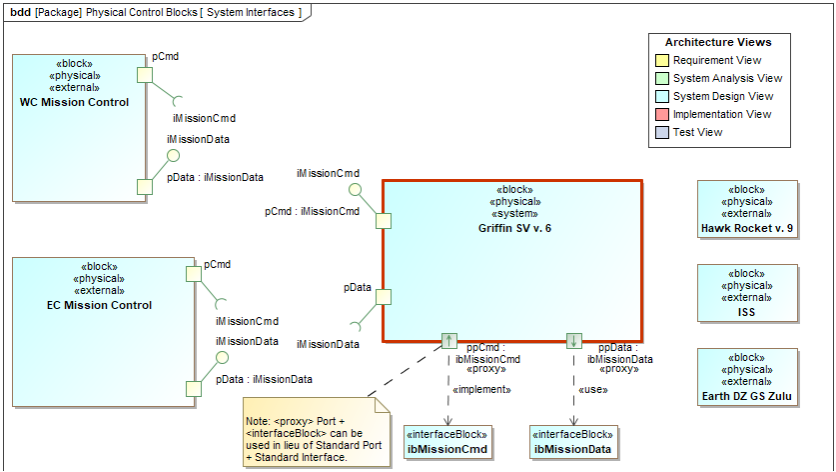
Exemple

BankAccount {balance $\geq$ lowest}
deposit(amount: Money) {*pre* : amount $>$ 0} {*post* : balance = balance@*pre* $-$ amount} withdraw(amount: Money) {*pre* : amount $>$ 0 $\wedge$ amount $\leq$ balance $-$ lowest} {*post* : balance = balance@*pre* $+$ amount}
Lifecycle = init .(deposit + withdraw)*.close
ResponseTime(deposit) $<$ 1*s* when *Users* $<$ 1000 *ResponseTime*(withdraw) $<$ 1*s* when *Users* $<$ 1000 Availability(BankAccount) all days from 1:00 to 0:00

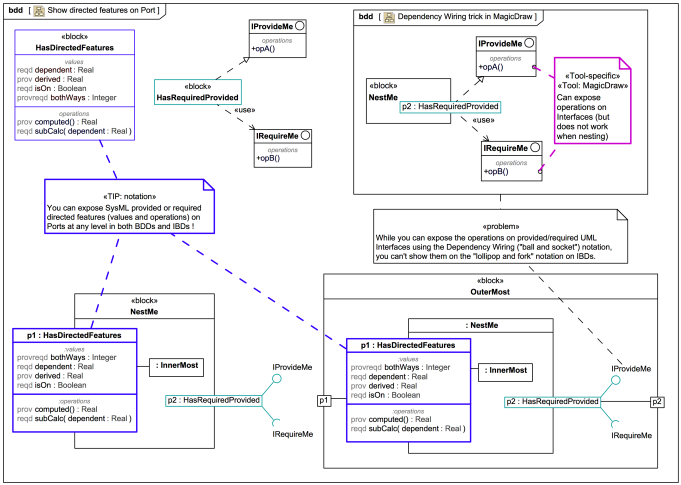
Expression des contrats

- Contrat syntaxique : langages - types (IDL, . . .)
- Contrat sémantique : OCL, assertions, Eiffel, JML
- Contrat de synchronisation : Automates, Protocol State Machine, logiques temporelles, . . .
- Contrat de qualité de service : QML

Notations SysML I



Notations SysML II



Conclusion sur les interfaces

- L'interface est un contrat : statique et dynamique
  - Essentiel pour spécifier des systèmes – intermédiation
  - Facilite l'utilisation – mode d'emploi
  - Facilite l'assemblage
  - Contraint/guide la conception
  - Permet la vérification
- La notion d'interface est au cœur de l'ingénierie système
- La notion d'interface est au cœur de l'étude des interactions

Avancement

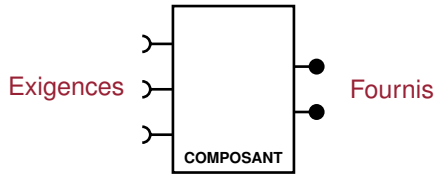
Définitions

- Un modèle d'architecture est composée de
  - composants (des boîtes)
  - connecteurs (des traits)

  pour former une configuration
- Architecture = structure avec des éléments et des relations

Composant

- Unité logique (pas forcément de déploiement)
- Qui possède une description explicite + ou - formelle (doc, contrat ...)
  - De ce qu'elle fait
  - De ce dont elle a besoin
- Et qui peut être assemblée via des ports



exemple de Java, méthodes publiques = F, import, une partie de E

Que représente un composant ?

- Diverses *unités*
  - un système
  - une fonction, une procédure, un calcul
  - un objet
  - un service
  - une unité de stockage
  - une interface
  - ...
- Un bon guide est d'y associer une responsabilité

Connecteur

- une connexion qui relie
- deux (ou plusieurs) composants qui
- jouent un role (dans la connexion)

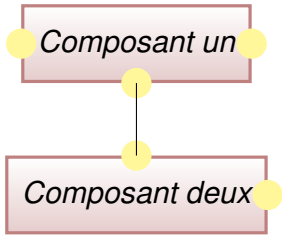
- ex : émetteur / recepteur, tcp, udp, bus
- peut relier plein de composants

Que représente un connecteur ?

- Divers “moyens de connection” . . .
  - un bus
  - un protocole
  - un appel de procédure (à distance)
  - . . .
- Un bon guide est d’y associer un transfert de donnée ou de contrôle

Une configuration

- Un ensemble de composants et de connecteurs assemblés
- Les ports sont associés à des rôles

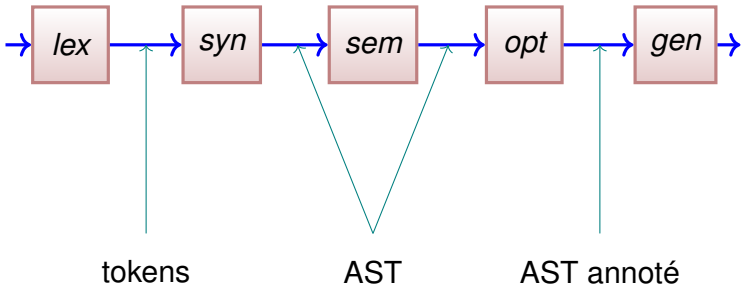


Quelle est l'architecture d'un compilateur ?

- Exemple issu de [GS+93]
- On identifie
  - des fonctions, des modules
  - des interactions
  - des données (table, arbre)
  - des entrées, des sorties
- Comment abstraire ?

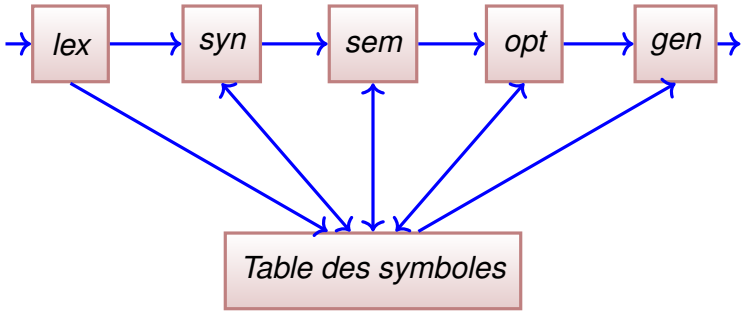
Un compilateur – architecture ’70

Une séquence de fonctions. . .
Et les erreurs



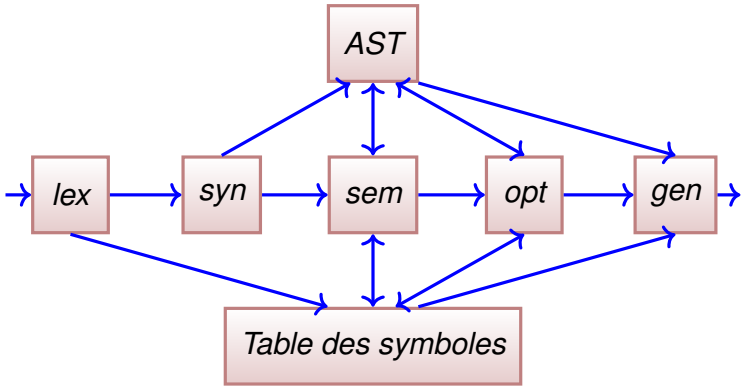
Un compilateur – architecture fin ’70

Une séquence de fonctions qui partagent une table de symboles. . .



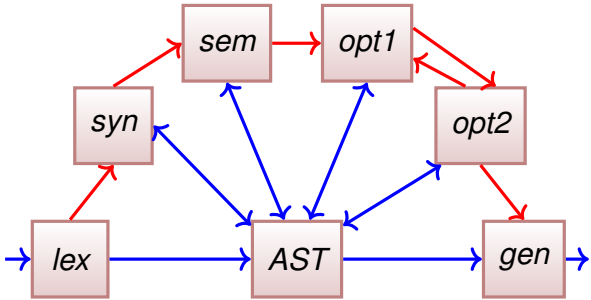
Un compilateur – architecture ’80

Une séquence de fonctions qui partagent une table de symboles et un arbre syntaxique abstrait (AST)…



Un compilateur - architecture ’90

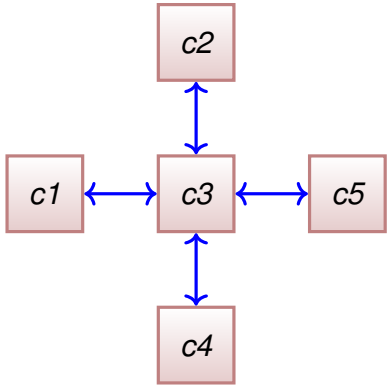
Une séquence de fonctions qui partagent des informations et enchaine des traitements optionnels. . .

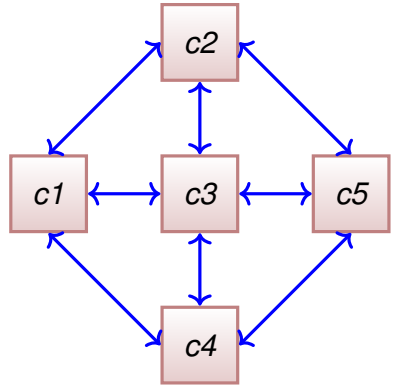


Un compilateur - synthèse

- On peut déjà dire beaucoup de choses avec la topologie : partage, goulot d'étranglement, point de défaillance ...
- Plusieurs styles
  - batch
  - tableau noir
- Des propriétés différentes : efficacité, évolutivité
- propriétés d'un assemblage $\neq$ somme des propriétés des éléments, certaines emergent (lié à l'assemblage)

Un autre exemple : comparons !

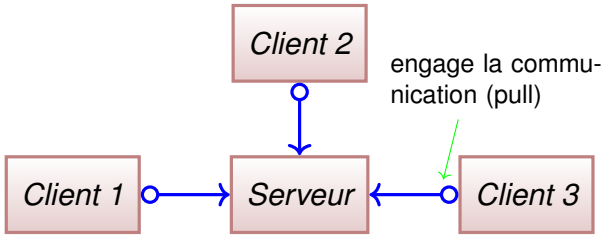




Avancement

Style d'architecture

- Ensemble de types de composants et de types de connecteurs
  - ex. Client, Serveur
- Règles d'assemblages à respecter

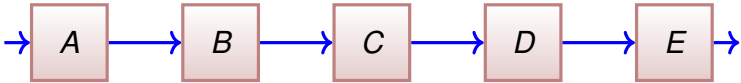


Catalogue de style

Design pattern d'architecture

- Dataflow
- Pipe & Filter
- Tableau noir
- Publish-subscribe
- En couches
- Client-Serveur
- Pair-à-pair
- N-tiers

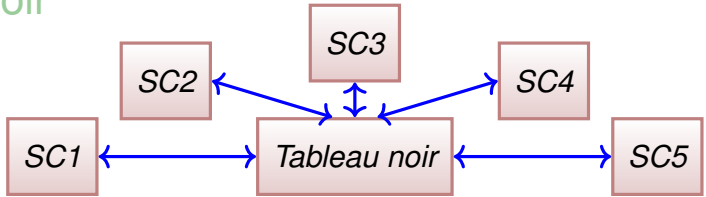
Dataflow / pipeline



Composants	*Connecteurs*
Filtre	Transfert de donnée mono-directionnel

Avantages	*Inconvénients*
Composition simple Couplage faible Boites interchangeables	1 entrée, 1 sortie seulement Plus simple avec des données homogènes

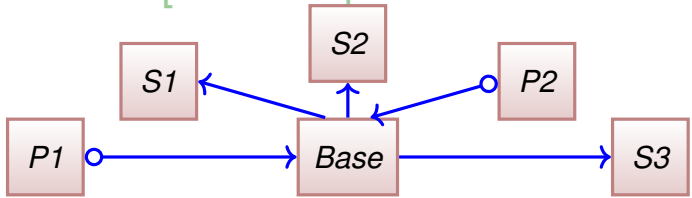
Tableau noir



Composants	*Connecteurs*
Tableau noir Source de connaissance	protocole bi-directionnel

Avantages	*Inconvénients*
Correction des erreurs rapide Peu de perte information Extensible Couplage faible Composition simple	Risque d'accumulation de données inutiles Contraintes de vocabulaire Dépendant du domaine d'application

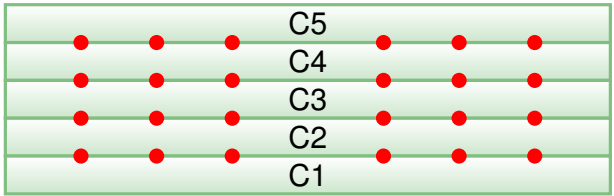
Publish-subscribe [EFGK03]



Composants	*Connecteurs*
Base Publisher Subscriber	protocole orienté message

Avantages	*Inconvénients*
Anonymat Couplage faible Extensible	Performance Asynchrone (?)

En couches



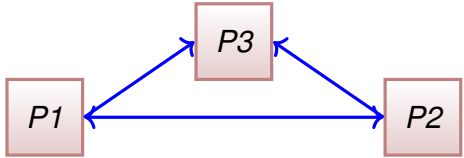
Composants	*Connecteurs*
Couche	Appel de procédure

Avantages	*Inconvénients*
Conception incrémentale Maintenance Réutilisation	Taille des niveaux Efficacité

Client-Serveur

Composants	*Connecteurs*
Serveur Client	protocole *pull*
Avantages	*Inconvénients*
Composition simple Couplage faible Extensible Efficace	Point à point seulement

Pair-à-pair



Composants	*Connecteurs*
Pair	protocole dédié
Avantages	*Inconvénients*
Symmétrie Résistance aux pannes Extensible	Performance

N-tiers

- Une variante du “client-serveur” ou de “en couches”
- Un tiers pour une responsabilité
- Exemple
  1. Tiers de persistance
  2. Tiers métier
  3. Tiers de présentation

Mélange de styles

- Un système complexe utilise plusieurs styles
  - Horizontalement, à un niveau donné, une partie du système peut être d'un style et une autres partie d'un autre style
  - Verticalement, quand on étudie la structure interne d'un composant d'un style, on peut y trouver un autre style
- Le découpage peut être plutôt technique ou métier

Avancement

Architecture Client Serveur

- Client : produit des demandes de service
- Serveur : attend des demandes de service

- Coordonner plusieurs demandes de services pour un client : session
- Coordonner plusieurs client : orchestration

Responsabilités des clients

- Interagissent avec l'utilisateur (IHM/contrôle)
- Peuvent faire des calculs (avec des informations locales ou collectées)
- Sollicitent des serveurs via un canal (middleware, protocole ...)
- Recoivent des réponses et les présentent à l'utilisateur (IHM/présentation)

Responsabilités des serveurs

- Attendent des sollicitations via un canal (middleware, protocole ...)
- Réalisent des calculs
- Gèrent les défaillances des services

Rappels sur HTTP

- Le protocole s'appuie sur 3 principaux verbes :
  - GET ; accède, retourne 200 (OK), 400 (Bad Request) ou 404 (Not Found)
  - POST ; crée, retourne 201 (Created), 404 (Not Found) ou 409 (Conflict)
  - PUT ; modifie/remplace, retourne 200 (OK), 404 (Not Found), 204 (No Content) ou 405 (Method Not Allowed)
- Il en existe d'autres
  - PATCH ; modifie partiellement, retourne 200 (OK), 404 (Not Found), 204 (No Content) ou 405 (Method Not Allowed)
  - DELETE ; détruit, retourne 200 (OK), 404 (Not Found) ou 405 (Method Not Allowed)

GET

- Retourne une représentation XML, JSON ou HTTP (200)
- Ne dois pas changer l'état du service/serveur
- Propriétés
  - Sûr
  - Idempotent ($f \circ f = f$)
- Exemples
  - `GET http://www.example.com/customers/12345`
  - `GET http://www.example.com/customers/12345/orders`
  - `GET http://www.example.com/buckets/sample`

POST

- Fabrique un nouvel ID (URI)
- Propriétés
  - Non sûr
  - Pas idempotent ($f \circ f \neg f$) ; risque de doublons
- Exemples
  - `POST http://www.example.com/customers`
  - `POST http://www.example.com/customers/12345/orders`

PUT

- Remplace un objet (URI) avec le corps (BODY) transmis
- Peut-être utilisé pour créer un objet si on transmet un ID (URI)
- Propriétés
  - Non sûr (car modifie l'état du serveur/service)
  - Idempotent ($f \circ f = f$) [Intentionnellement, car le service peut rendre un PUT non idempotent]
- Exemples
  - `PUT http://www.example.com/customers/12345`
  - `PUT http://www.example.com/customers/12345/orders/98765`
  - `PUT http://www.example.com/buckets/secret_stuff`

HTTP protocol

- L'automate qui décrit un cycle de traitement d'une requête HTTP
- `https://github.com/kbrw/ewebmachine/blob/master/assets/http_diagram.png`
  - La chaine des cas d'erreur (donc des priorités)
  - Le filtre des langages
  - La gestion des caches (bas, milieu droit)
  - La gestion des verbes (angle haut-droit)

Conclusion

- Une architecture définit une topologie d'interaction et contraint la construction
- Elle montre (ou peut montrer)
  - la topologie
  - les couplages
  - les lieux des interfaces
  - le sens des échanges
  - les initiatives des échanges
- Elle cache
  - le détail des interfaces
  - l'ordre des échanges (aspect temporel)
- Pour aller plus loin : `https://www.youtube.com/watch?v=DngAZyWMGR0`

References I

Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec.
The many faces of publish/subscribe.
ACM computing surveys (CSUR), 35(2) :114–131, 2003.

David Garlan, Mary Shaw, et al.
An introduction to software architecture.
Advances in software engineering and knowledge engineering, 1(3.4), 1993.