Architecture

Gerson Sunyé
gerson.sunye@univ-nantes.fr
http://sunye.free.fr

Architecture

- Abstraction de l'implémentation du système: choix stratégiques.
- Définition des parties composantes, des caractéristiques externes de chaque partie et des relations entre ces parties.
- Ensemble de décisions et de contraintes: e.g. utiliser CORBA, MVC, trois-tiers, getX() et setX(), etc.

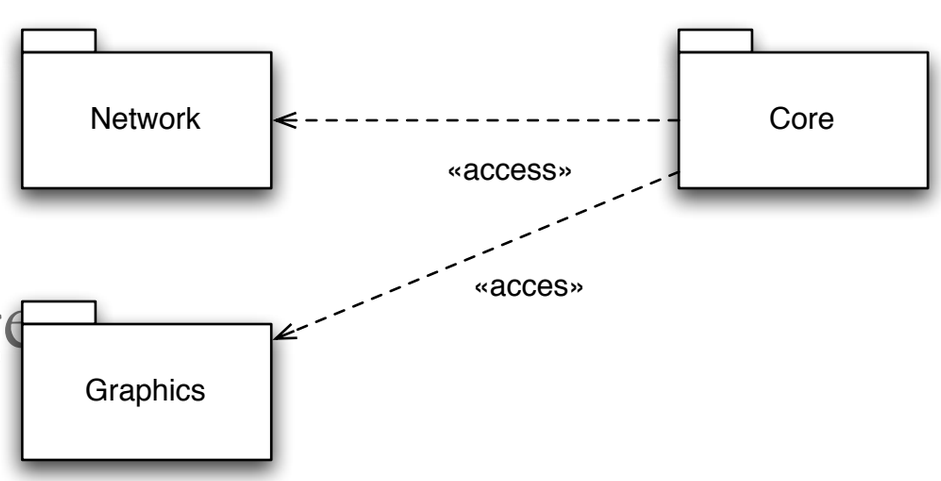
Vues architecturales

- Application (logique).
- Processus.
- Déploiement (technique).
- Fiabilité et sécurité.

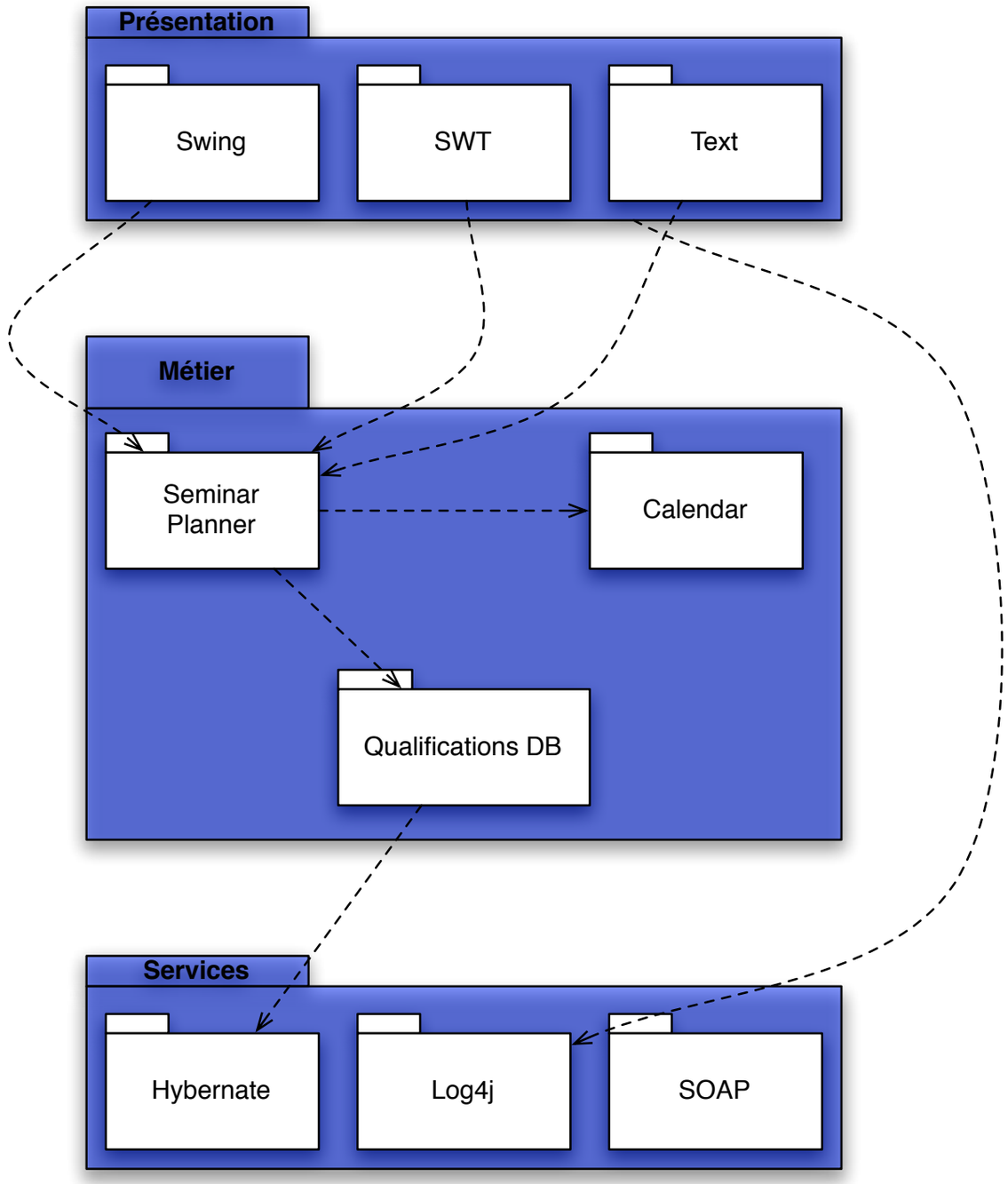
Architecture logique

Architecture logique

- Organisation de l'application.
- Relations de dépendance entre package



Exemple


Présentation
Swing
SWT
Text
Métier
Seminar Planner
Calendar
Qualifications DB
Services
Hybernate
Log4j
SOAP

Patrons architecturaux:

- Layers.
- Pipes and filters.
- Autres: Trois-tiers, PAC, MVC, SOA, Blackboard, Naked objects, etc.

Architecture technique

Architecture technique

- Description du déploiement de l'application sur les différents noeud logiques.
- [Rappel] UML - Diagramme de déploiement.

Broker

[Douglass 02]

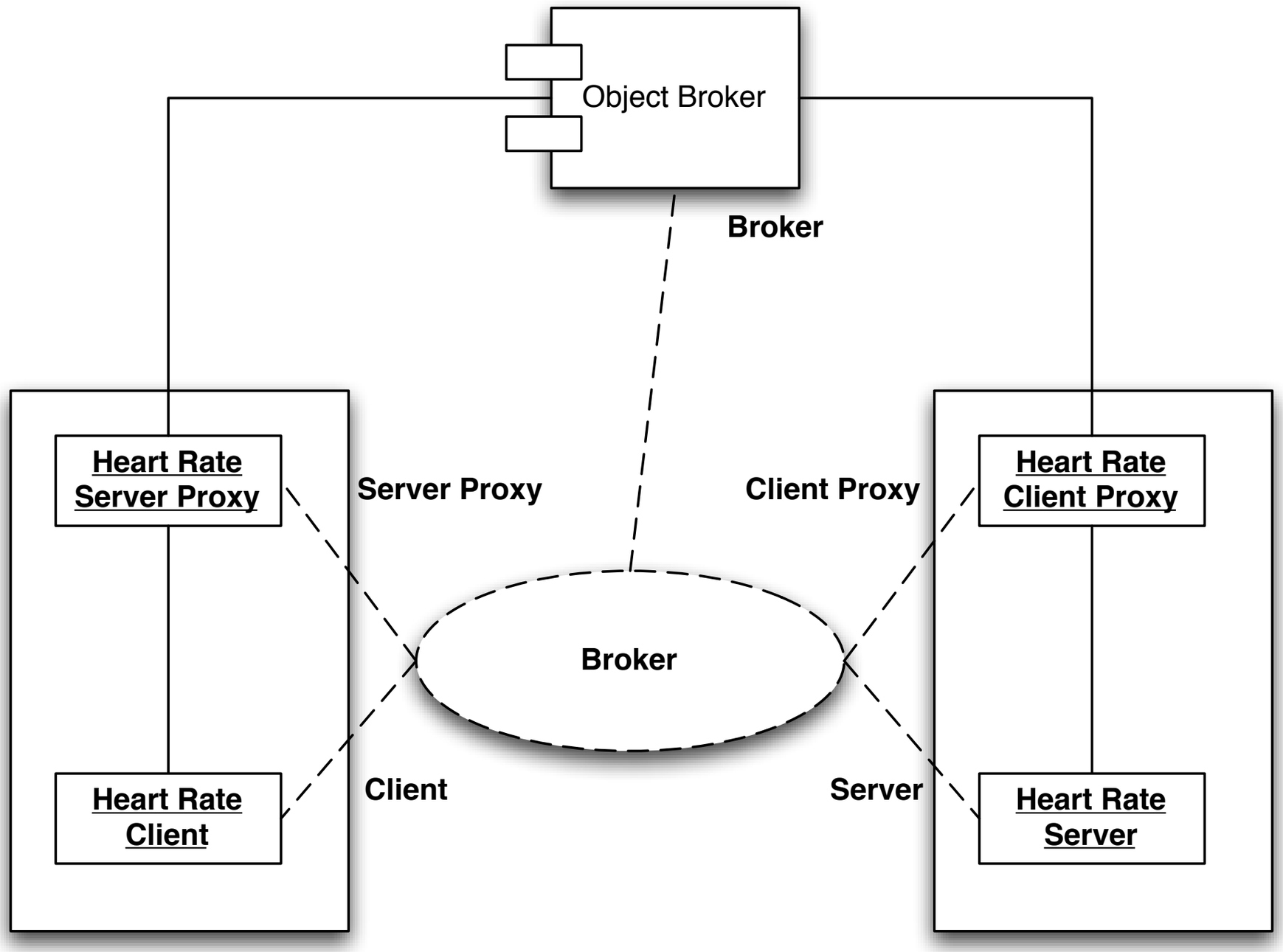
Objectif

- Permettre l'utilisation d'un objet distant sans connaître son emplacement pendant la compilation

Motivation

- C’est une version élaborée d’un Procureur (Proxy), qui découple les clients des serveurs.

Structure


Object Broker
Broker
Heart Rate
Server Proxy
Server Proxy
Client Proxy
Heart Rate
Client Proxy
Broker
Heart Rate
Client
Client
Server
Heart Rate
Server

Conséquences

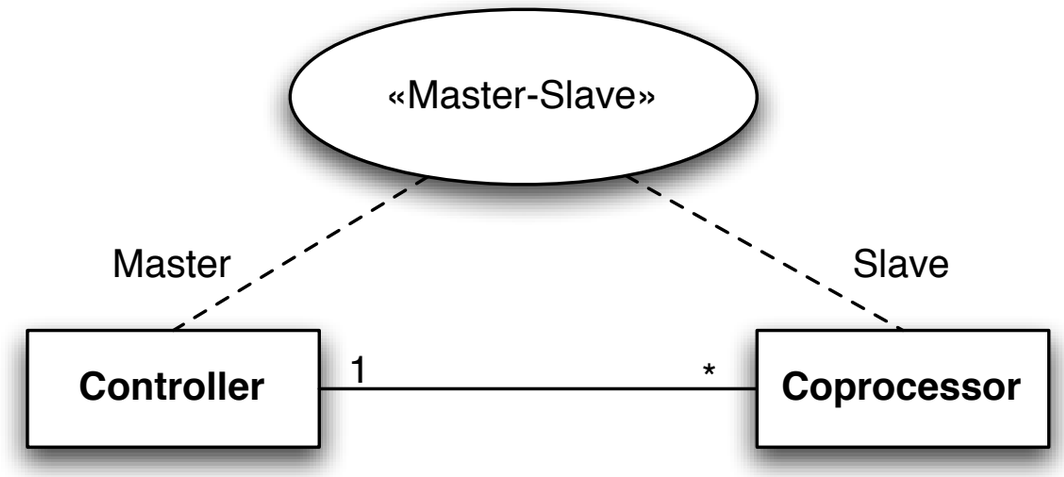
- L'emplacement du serveur n'est connu que lors de la connexion au courtier
- Le Courtier peut maintenir l'intégrité d'un système tolérant aux failles ou critique, en employant un chien de garde (watchdog)

Master-Slave

[Buschmann96]

Master-Slave

- Un *maître* distribue le travail à un ensemble d'*esclaves*
- Distribution
  - Statique (à la compilation)
  - Dynamique (répartition de charge)
  - Redondante (tolérance simpliste aux failles)



Autres patrons

- Distribution patterns [Douglass02]:
  - Shared Memory
  - Remote Method Call
  - Data Bus

Processus

Architecture des processus

- Mode de contrôle
- Spécification de la concurrence

Contrôle

- Procédural, traitement par lots (*batch*)
- Événementiel
- Concurrent
- Autres

Concurrence

- Intrinsèque
  - Imposé par l'environnement: interfaces multiples, etc.
- Choix d'ingénierie
  - Traitements en parallèle
  - Redondance

Objets actifs

- Objet qui commence à exécuter son comportement lors de sa création et continue à le faire jusqu'à sa destruction (ou à la fin de son comportement)

Implémentation d'objets actifs

- Approche naïve:
  - Allouer un processus à chaque objet actif.
- Inconvénients:
  - *Context Switching Overhead*
  - Ressources système gaspillées
    - Processus inactifs en attente d'événements

Structures de données

- Objets atomiques.
- File à priorités.
- File bloquante.
- Tableau associatif concurrent.
- Pool de threads.

Reactor

[Schmidt95]

Objectif

- Ce patron traite des demandes de services concurrentes faites par un ou plusieurs clients.

Contexte

- Une application serveur d'un système réparti qui reçoit des événements concurrents d'un ou de plusieurs clients.

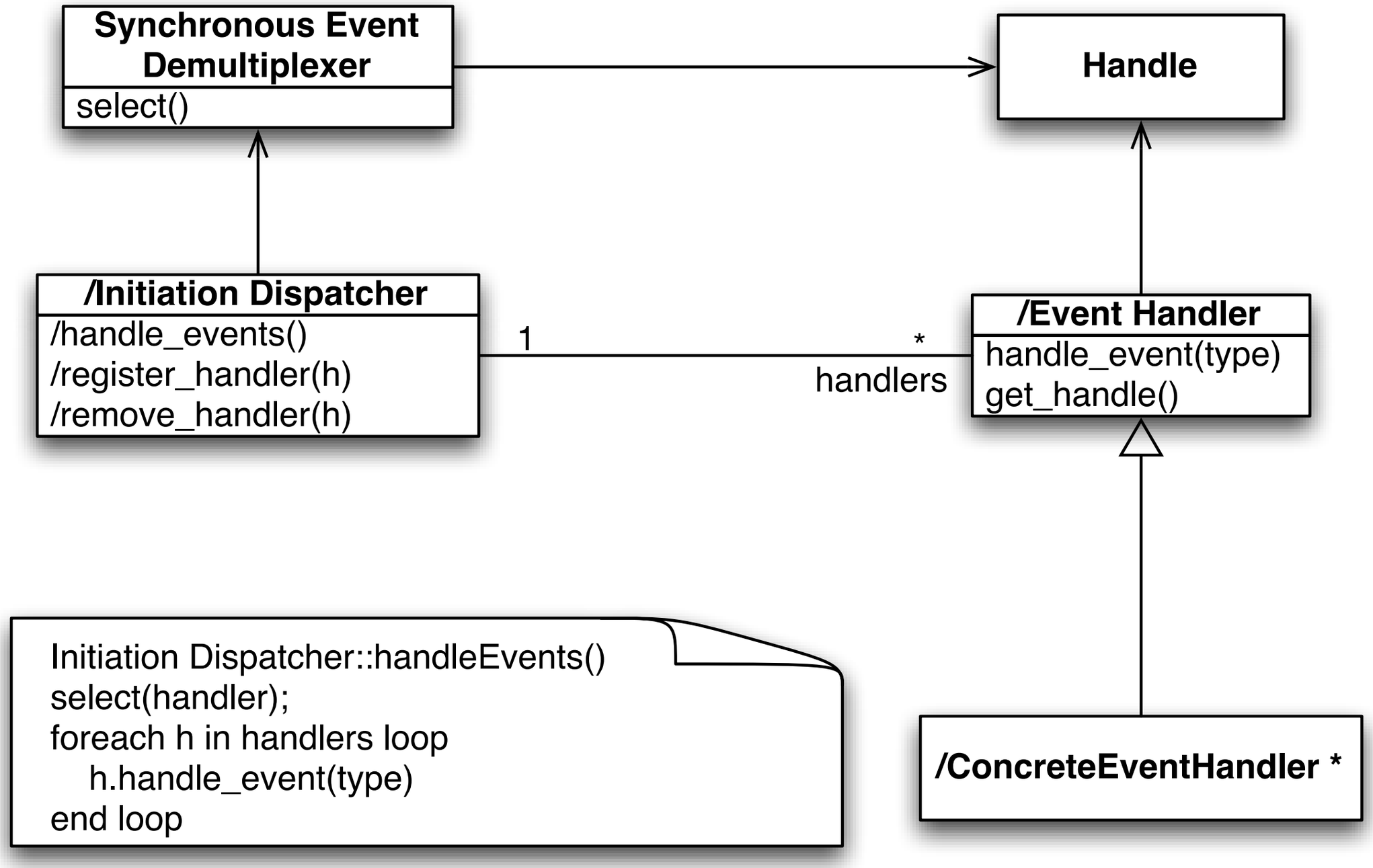
Forces

- Disponibilité: Le serveur doit être disponible pour traiter les demandes.
- Efficacité: Le serveur doit minimiser le temps de latence, maximiser la capacité de traitement et éviter d'utiliser le processeur sans nécessité.
- Simplicité: Le serveur doit être conçu de manière à simplifier l'utilisation de différentes stratégies de concurrence.
- Adaptabilité: La création de nouveaux services ne doit pas changer le format des messages.

Solution

- Utiliser un *dispatcher* d'événements au coeur du système.

Structure


Synchronous Event Demultiplexer
select()
Handle
/Initiation Dispatcher
/handle_events()
/register_handler(h)
/remove_handler(h)
1
*
handlers
/Event Handler
handle_event(type)
get_handle()
Initiation Dispatcher::handleEvents()
select(handler);
foreach h in handlers loop
h.handle_event(type)
end loop
/ConcreteEventHandler *

Conséquences

- Non pré-emptif
- Difficile à déboguer

Utilisations

- InterViews
- ACE
- CORBA ORB
- Apache XML Cocoon

Message Queuing

[Douglass02]

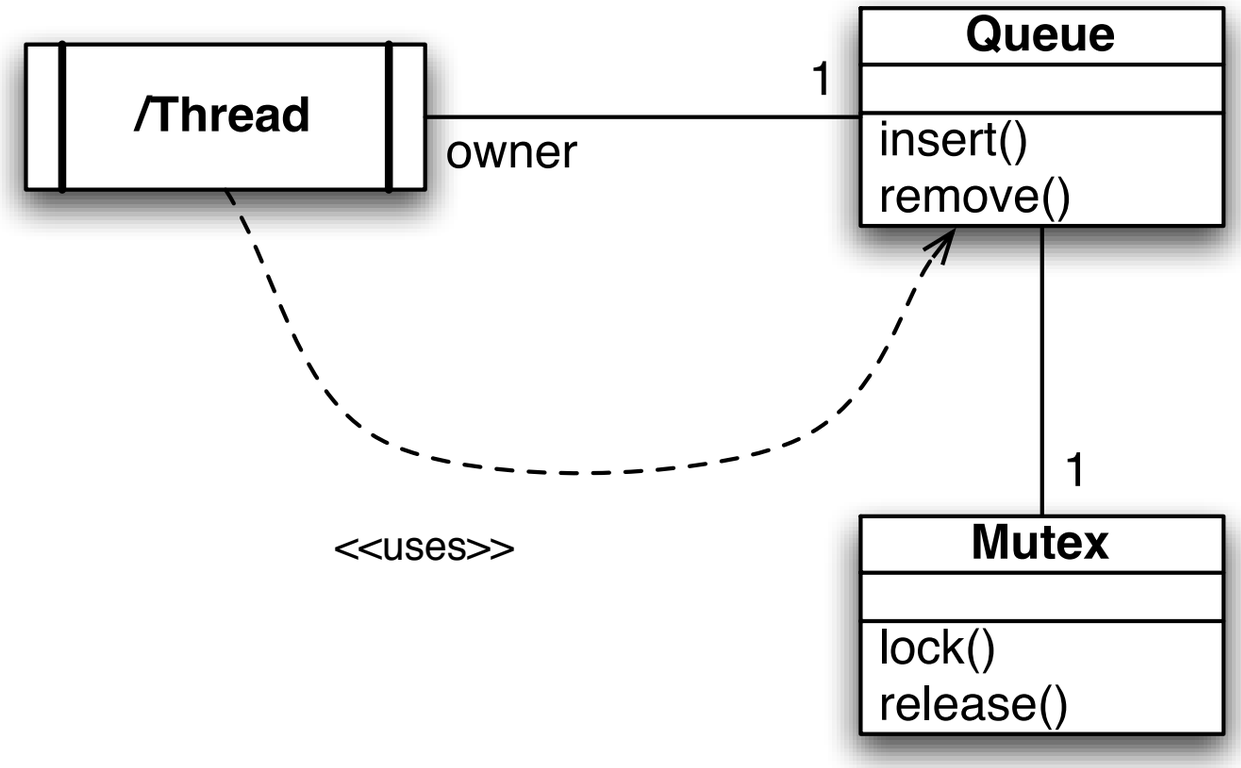
Objectif

- Fournir un mécanisme simple d'échange d'informations entre *threads*.
- Prendre en compte le problème de l'exclusion mutuelle.

Contexte

- Dans les systèmes multi-thread, les threads doivent:
  - Synchroniser pour échanger des informations
  - L'échange d'information doit éviter les situations de compétition (*race conditions)*

Structure


/Thread
owner
1
Queue
insert()
remove()
1
<<uses>>
Mutex
lock()
release()

Conséquences

- Mécanisme simple, utilisé par la plupart des systèmes temps-réel.
- Relativement lourd.

Autres patrons

- Concurrency patterns [Douglass02]
  - Concurrency
  - Interrupt
  - Guarded Call
  - Rendezvous
  - Cyclic Executive
  - Round Robin
  - Static Priority
  - Dynamic Priority

Fiabilité et sécurité

Fiabilité et sécurité

- Spécification de la redondance
- Prévision des conditions limites:
  - Initialisations
  - Terminaisons
- Gestion de défaillances et de reprises.

Gestion de défaillances

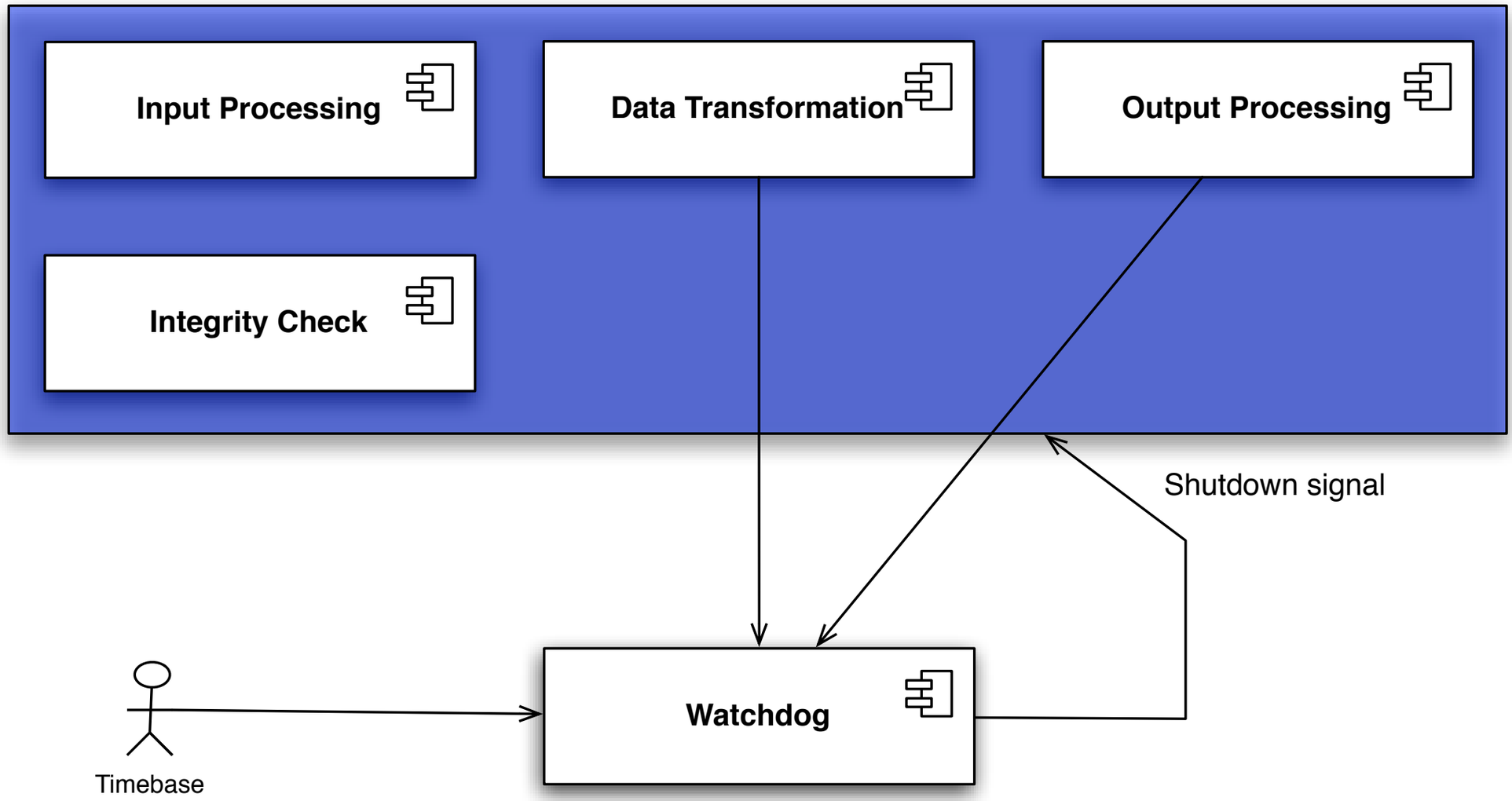
- Typiquement:
  - Réessayer l'opération
  - Utiliser des informations redondantes pour corriger la défaillance
  - Aller à un état *fail-safe*
  - Alerter quelqu'un
  - Redémarrer le système

Watchdog

Objectif

- Assurer qu'un processus dépendant du temps marche correctement

Structure


Input Processing
Data Transformation
Output Processing
Integrity Check
Shutdown signal
Watchdog
Timebase

Conséquences

- Trop simple pour être utilisé seul dans un système critique.
- Bien adapté pour déterminer des défaillances de temps, ou des inter-blocages (deadlocks).

Autres patrons

- [Douglass02]:
  - Protected Single Channel
  - Homogeneous Redundancy
  - Triple Modular Redundancy
  - Heterogeneous Redundancy
  - Monitor-Actuator

- Sanity Check
- Safety Executive

Autres patrons

Proxy

[GoF]

Observer

[GoF]

Architecture

Gerson Sunyé

gerson.sunye@univ-nantes.fr