Tecniche di Progettazione: Design Patterns

GoF: Mediator & Coordination

Applicability

- A set of objects communicates in well-defined, but complex ways. Often with unstructured dependencies.
- Reusing objects is difficult because it refers to and communicates with many other objects.
- A behavior that's distributed between several classes should be customizable without a lot of subclassing.

ATC Mediator

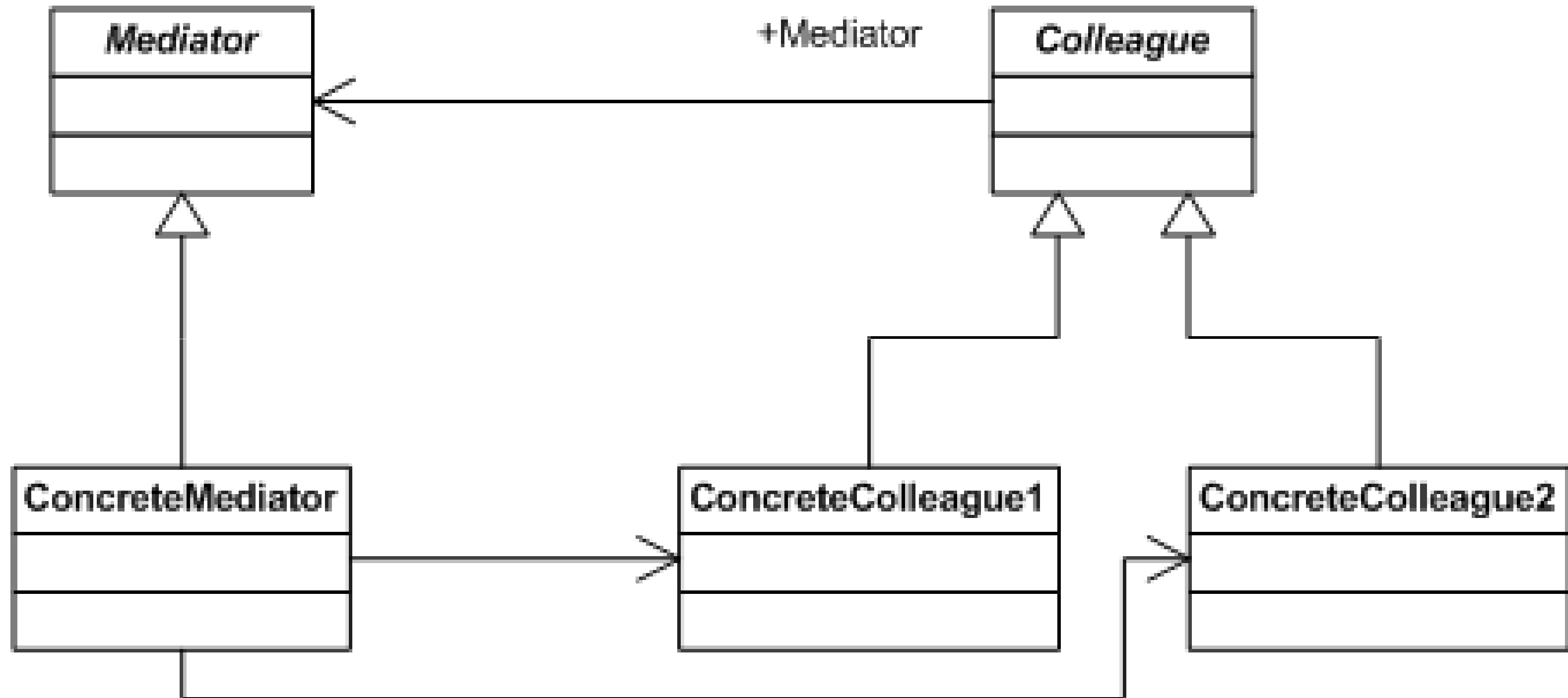
Flight 111

Flight 1011

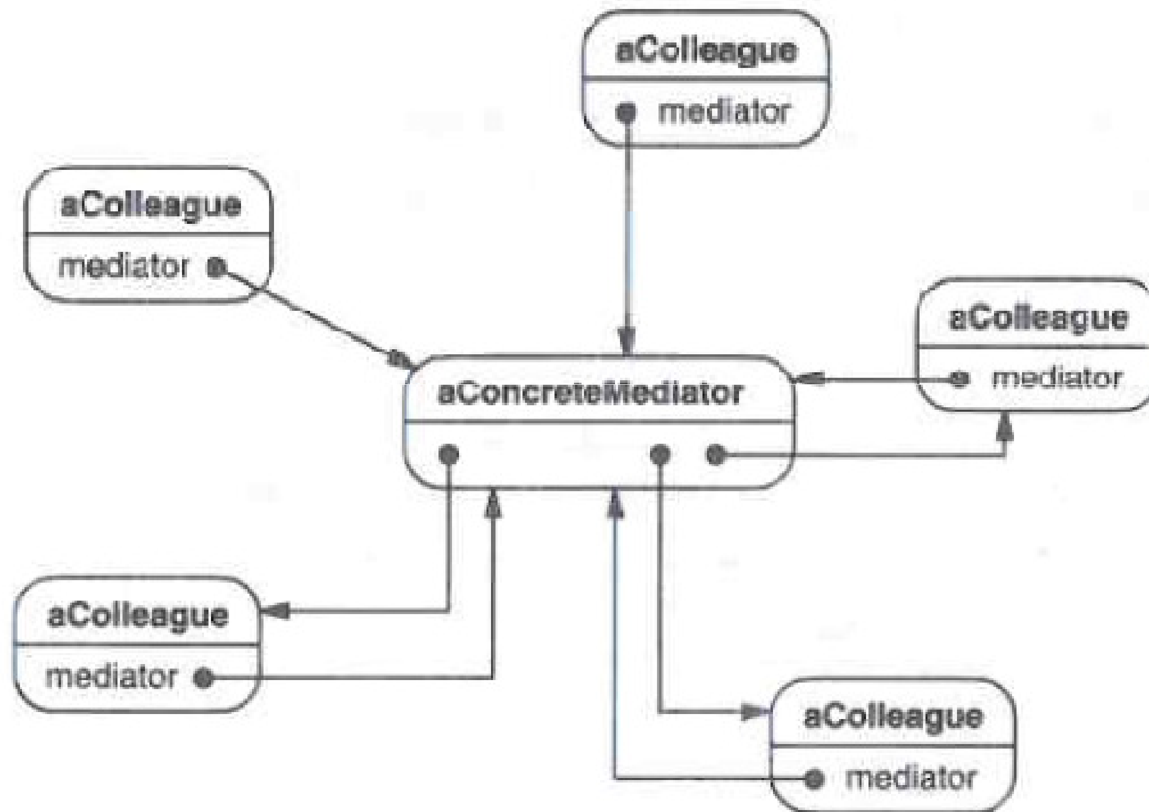
Flight 112

Flight 747

Mediator: structure



Structure



Mediator

- A mediator acts as a HUB of communication for a group of interdependent objects.
- In other words, a mediator object is the sole object that knows all other objects in a group of collaborating objects and how they should interact with each other.
  - All other objects should interact with the mediator object, instead of each other.

Consequences

- Decouples colleagues
  - Colleagues become more reusable.
  - You can have multiple types of interactions between colleagues, and you don't need to subclass or otherwise change the colleague class to do that.
- Limits subclassing
  - Localizes behavior that would be otherwise distributed among many objects
  - Changes in behavior require changing only the Mediator class

Consequences

- Simplifies object protocols
  - Many-to-many interactions replaced with one-to-many interactions
  - More intuitive
  - More extensible
  - Easier to maintain
- Abstracts object cooperation
  - Mediation becomes an object itself
  - Interaction and individual behaviors are separate concepts that are encapsulated in separate objects

Consequences

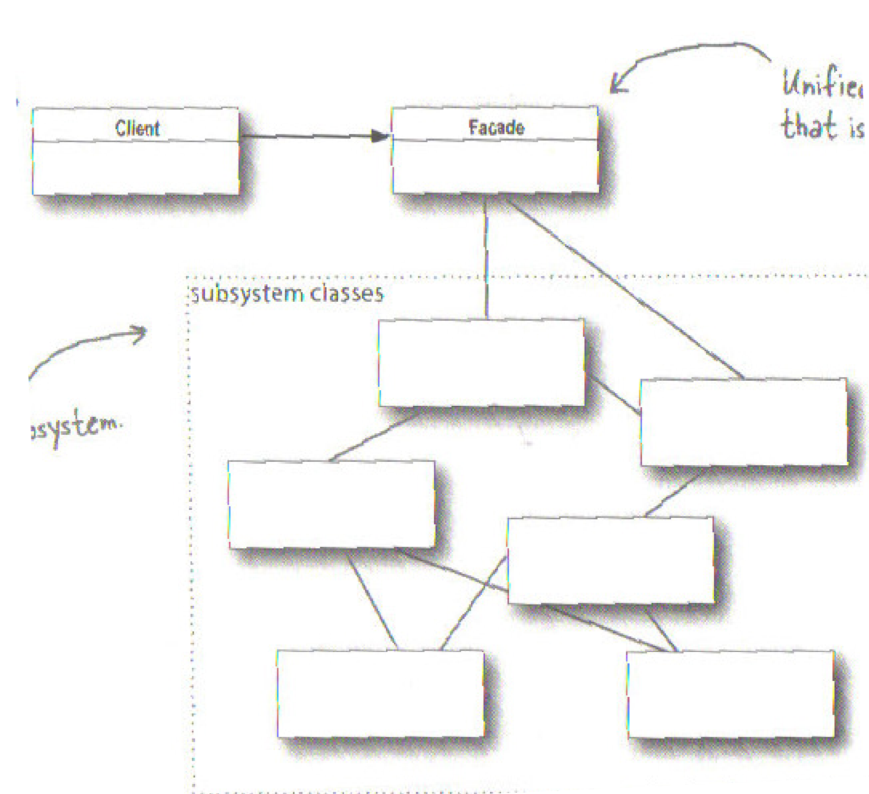
- Centralizes control
  - Mediator can become very complex
  - With more complex interactions, extensibility and maintenance may become more difficult
  - Using a mediator may compromise performance

Implementation Issues

- Omitting the abstract Mediator class – possible when only one mediator exists
- Strategies for Colleague-Mediator communication
  - Observer class
  - Pointer / other identifier to "self" passed from colleague to mediator, who pass it to the other colleague(s)

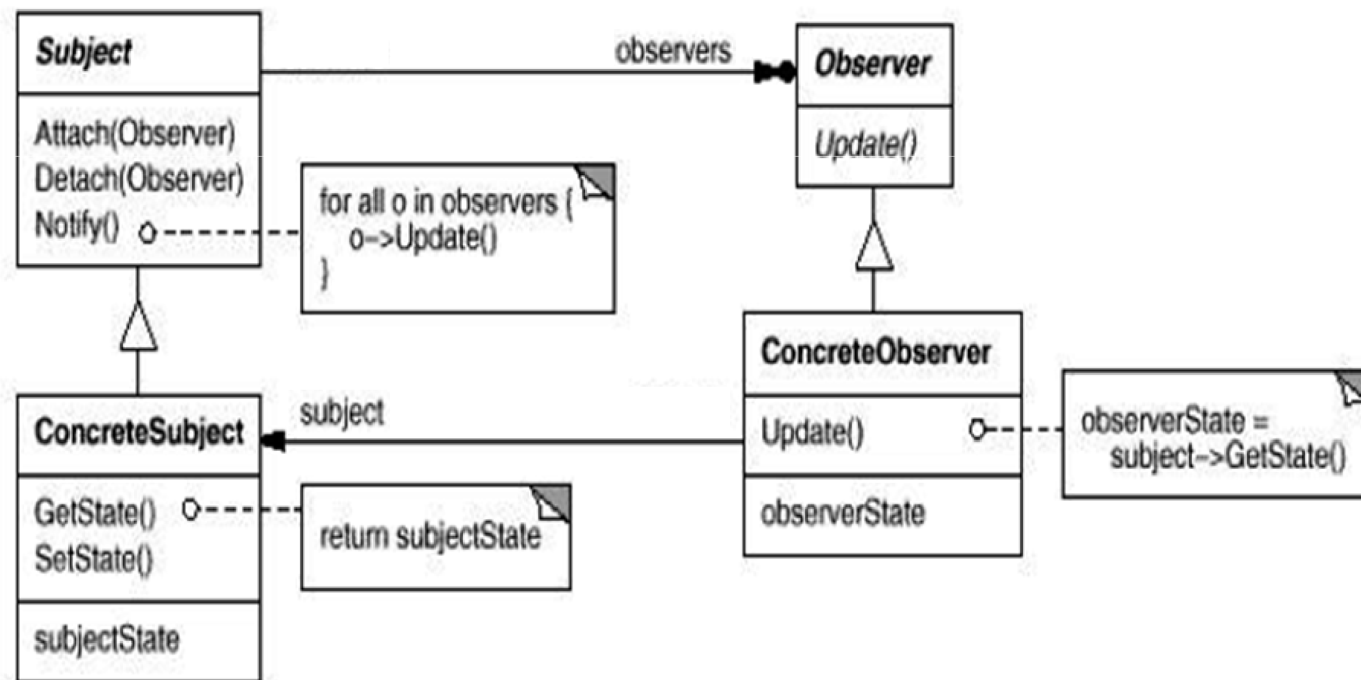
Related Patterns: Façade

- Unidirectional rather than cooperative interactions between object and subsystem
- Mediator is like a multi-way Façade pattern.
- A facade is a "unified interface" for a set of interfaces in a subsystem - for use by consumers of the subsystem - not among the components of the subsystem.



Related Patterns: Observer

- May be used as a means of communication between Colleagues and the Mediator

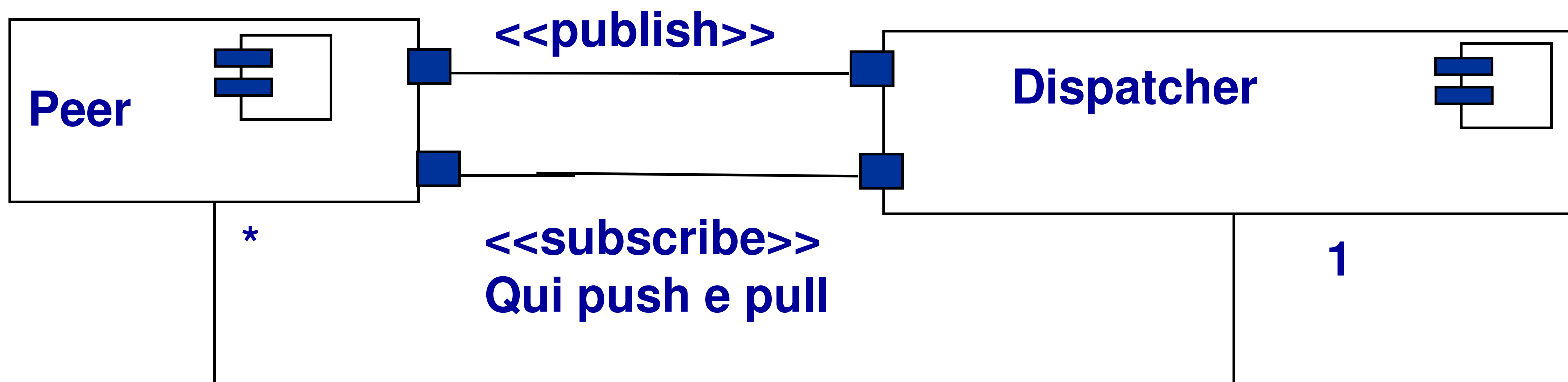


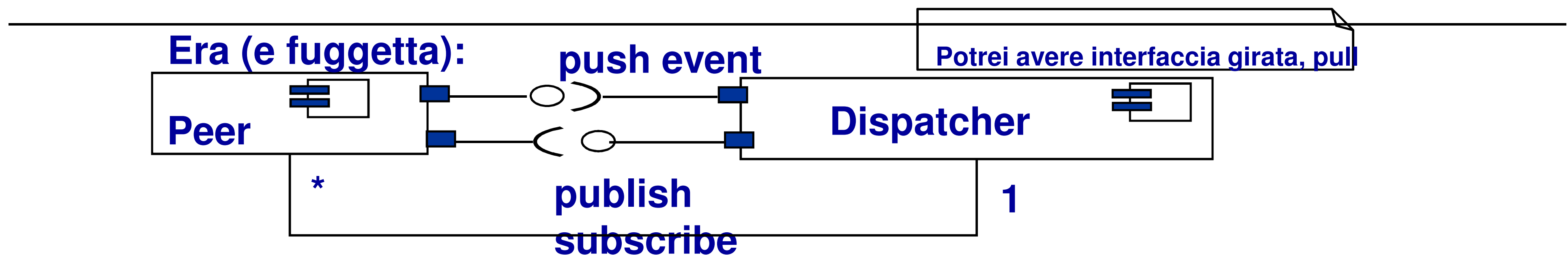
Pattern architetturale: Publish-subscribe

- Le componenti interagiscono annunciando eventi: ciascuna componente si "abbona" a classi di eventi rilevanti per il suo scopo
- Ciascuna componente, volendo, può essere sia produttore che consumatore di eventi
- Disaccoppia produttori e consumatori di eventi e favorisce le modifiche dinamiche del sistema

Pattern architetturale: Publish-subscribe

Operazioni: subscribe, unsubscribe, publish, notify (push), letMeKnow (pull)





Pattern architetturale: Publish-subscribe

- In questo stile, mittenti e destinatari di messaggi dialogano attraverso un tramite, detto dispatcher o broker.
- Il mittente di un messaggio (detto publisher) non deve essere consapevole dell'identità dei destinatari (detti subscriber); esso si limita a "pubblicare" (to publish) il proprio messaggio al dispatcher.
- I destinatari si rivolgono a loro volta al dispatcher "abbonandosi" (to subscribe) alla ricezione di messaggi.
- Il dispatcher quindi inoltra ogni messaggio inviato da un publisher a tutti i subscriber interessati a quel messaggio.

Coordination

- Coordination is managing dependencies between activities.
  - — Malone and Crowston, CACM, 26.1
- Blackboards
- Linda and tuple spaces
- BPEL (Business Process Execution Language) and web services (BPEL4WS o WS-BPEL)

Blackboard

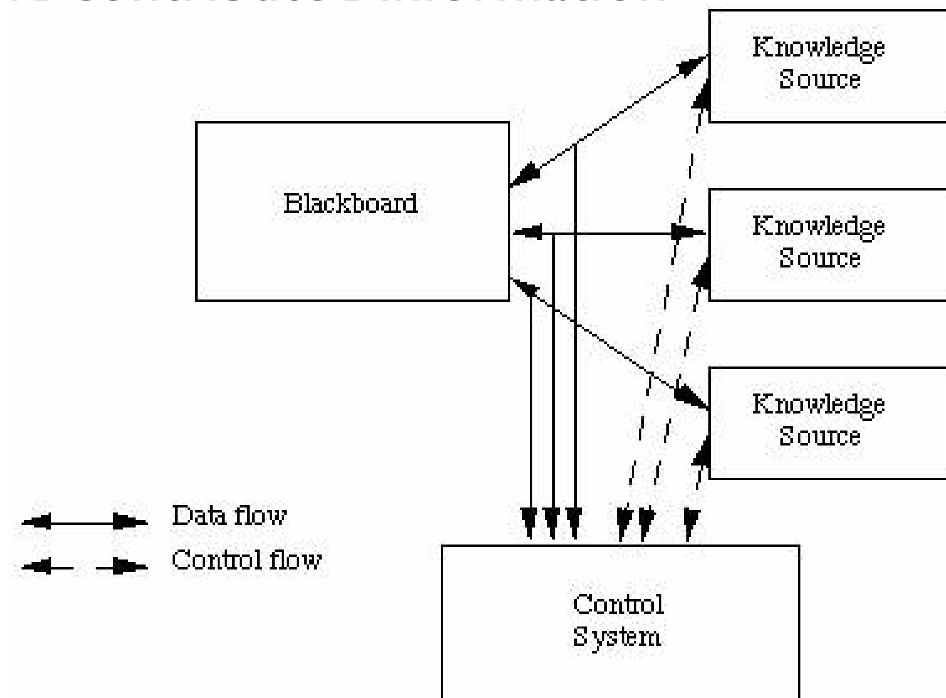
- Barbara Hayes-Roth
  A blackboard architecture for control.
  Artificial Intelligence, 1985, 26, 251-321.

- A shared data repository
  - stores all key system data

- Other components
  - communicate with each other via the blackboard
  - respond to changes in the state of the blackboard

Blackboard

- Blackboard systems consists of three major components:
  - The software specialist modules, called knowledge sources
  - The blackboard, a shared repository of problems, partial solutions, suggestions, and contributed information
  - The control shell, which controls the flow of problem-solving activity



Blackboard

- The **blackboard** approach provides freedom from message-passing constraints. The message-passing paradigm, although modular, requires a recipient of the message as well as a sender.

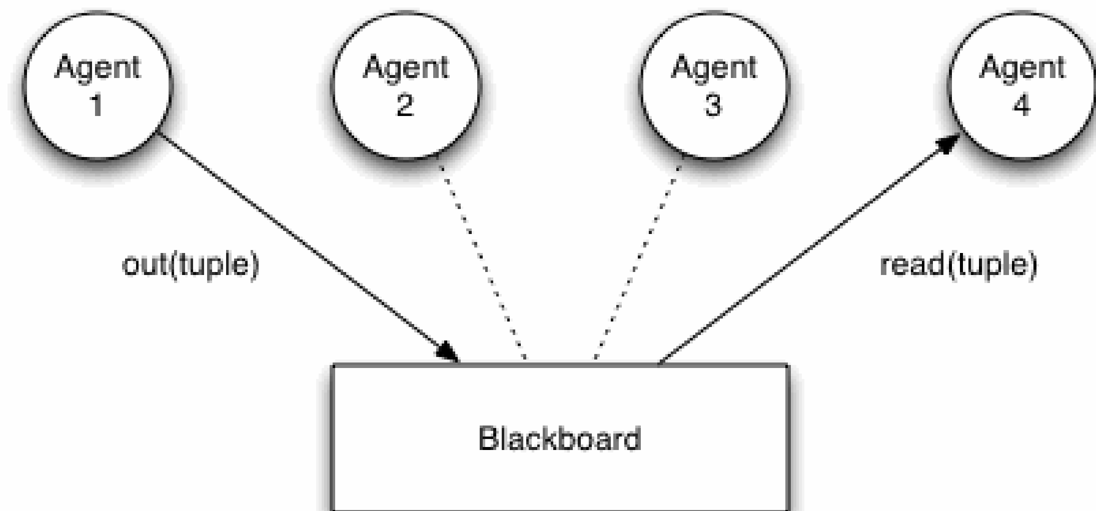
- Often, the recipient is not known, or the recipient might have been deleted. In the **blackboard** approach, the "message" is placed on the **blackboard**, and the developer of the module is freed from worrying about other modules"
  - --Nii, ***Blackboard*** *Architectures and Applications*

Blackboard

- The **Blackboard** pattern uses opportunistic reasoning in solving problems for which no deterministic solution strategies are known before hand, or for problems too vast for a complete exhaustive search
- The **Blackboard** pattern provides a design in which several specialized subsystems utilize their partial knowledge bases and strategies to build an approximated solution to the original problem.
- **Blackboard** problems are often solved with solutions involving highly recursive languages such as Lisp, or backward-chaining engines such as Prolog

Form blackboards to tuple spaces

- Blabkboards extended to non AI domains, e.g. as a communication media between distributed processes.
- Tuple spaces
- Tuple spaces were the theoretical underpinning of the Linda language David Gelernter & Nicholas Carriero at Yale University.



Problem

In concurrent and distributed programming we need mechanisms for controlling interactions between the *autonomous* components

- “Come and go as they please”
- Turn on/off
- Connect/Disconnect
- Plug in/Plug out
- Fail/Crash

Components

- Active Data Storage
  - Database + Coordination System
  - Stores data
  - Also tracks which agents are interested in what data
  - Notifies agents when data is available
    - Pushes notifications to agents
    - Agent clients don't need to poll
- Agent
  - "Clients" of the tuple space
  - Different from Client/Server
    - Agents contain all intelligence in the system
    - There is no application layer server code
    - aka Knowledge Sources

Connectors

- Data subscription
  - Agents can register interest in data
  - Subscription remains in affect until data is available
    - Critical difference compared to database queries
  - Provides temporal decoupling
    - Simplifying synchronization
- Data publication
  - All communication between agents done by modifying active data storage
  - Similar to traditional database updates except also trigger notification to subscribers

Example: Carnegie Mellon Uni. Robotics

- The CMU Navlab group builds robot cars, trucks, and buses, capable of autonomous driving or driver assistance
- Data Storage
  - information the Robot knows of its environment
- Agents
  - Robot parts/sensors
    - Actually software components that control robot parts
- Robot parts share data and react to changes of environment
  - … But parts do not directly depend on each other
  - … Some parts can fail or be destroyed without compromising the entire robot

Platforms

- Programming technology to implement blackboard-based systems
  - Linda
  - JavaSpaces
  - GigaSpaces
  - Apache River
  - Many more …
- We'll learn about Linda
  - A specialized Blackboard programming language
  - Active storage is called a "Tuple Space"
- Nice easy to read syntax

Linda

- Linda è un linguaggio di coordinamento che estende i linguaggi tradizionali permettendone l'utilizzo nello sviluppo di applicazioni in ambiente distribuito.
- Linda è indipendente dall'architettura sottostante (a memoria condivisa o rete di calcolatori) e dal linguaggio sequenziale usato (C, C++, etc …).
- Linda implementa una memoria associativa logicamente con divisa da tutti i processi dell'applicazione , detta Spazio delle Tuple.
  - Si parla di memoria associativa perché le tuple vengono identificate tramite matching su una chiave piuttosto che utilizzare un meccanismo tradizionale di indirizzamento.
  - .

Le tuple

- Una tupla è una sequenza di campi con tipo

  (“ linda ”, 2, 3.14)

  (sqrt(3.14), #FF3D0B,’t’)

  (0,’a’)

- Ogni campo può contenere un valore di un qualsiasi tipo semplice o strutturato) ammesso dal linguaggio sequenziale utilizzato.

Operazioni sullo spazio delle tuple

- out(t) eval(t) Generano nuove tuple
- in(s) Elimina tuple dallo spazio delle tuple
- rd(s) Legge tuple dallo spazio delle tupl

- Il parametro di out() e di eval () è una tupla t
- Il parametro di in() e di rd() è un' anti tupla (o template).
- Un'anti tupla è una sequenza di campi tipati che possono essere o dei campi valore (parametro attuale) o un campo indefinito (parametro formale) introdotto da "?"
- (?radix, ``a string'', ?j, 100)

Regole di Matching

- Si ha un matching tra una tupla *t* ed un'anti tupla *s* se:
- il numero di campi di *t* e di *s* è lo stesso
- si ha una corrispondenza tra i tipi di ogni campo
- i campi valore di *t* sono uguali ai campi valore di *s*

(?radix, ``a string'', ?j)

(sqrt(2), ``a string'', 100)

Tuple Space Operations

- Subscription Operators
- in(t)
  - find and remove a "matching" tuple from the tuple space; block until a matching tuple is found
- rd(t)
  - like in(t) except that the tuple is not removed

Tuple Space Operations

- Publication Operators
- out (t)
  - insert a tuple t into the Tuple Space

- eval(t)
  - send a live tuple into the tuple space which gets evaluated into a ‘data’ tuple
  - spawn/fork Agents in the Tuple Space
  - results of calls are collected and inserted as tuple

    eval(“f_calc ”, i, f(i, st))

Example

- N-element vector stored as n tuples in the Tuple space:

  ("V", 1, element1)

  ("V", 2, element2)

  …

  ("V", n, elementn)

- To read the jth element and assign it to x, use

  rd ("V", j, ?x)

- To change the ith element, use

  in ("V", i, ?OldVal)

  out("V", i, NewVal);

Example: Agent Synchronization

- Each Agent within some group must wait at a barrier until all Agents in the group have reached the barrier; then all can proceed.
- Set up barrier:

```
out ("barrier", n);
```

- Each Agent does the following:

```
in("barrier", ?val);
out("barrier", val-1);
rd("barrier", 0);
```

Semafori

- Operazioni sui Semafori
- out("semaphore") inizializzazione del semaforo
- in("semaphore") acquisizione del semaforo
- out("semaphore") rilascio del semaforo

- es . processo che usa un semaforo

  in("semaphore")

  sezione critica

  out("semaphore")

Dining Philosopher’s Problem

- N agents compete for use of N data resources (tuples)
- Agents are labeled 0 to N-1
- Agents require 2 resources in order to “eat()”
  - Agent 0 shares a resource with Agent 1 and Agent N
  - for i != 0
    - Agent i needs 1 resource that is needed by Agent i + 1 % N
    - Agent i needs 1 resource that is needed by Agent i - 1
- Agents release resources after “eat()”

Dining Philosophers Problem

```
initialize()
{
    int i;
    String[] d = {…..};
    for (i=0; i < num; i++) {
          out ("chopstick", i, d[i]);
    }
     for (i=0; i < num; i++) {
               eval ( phil(i) );
    }
}
```

```
void phil(int i) {
   while (true)  {
    String d1, d2;
    in("chopstick", i, ?d1);
    in("chopstick", (i + 1) % n, ?d2);
     eat (d1, d2);
     out("chopstick", i, d1);
     out("chopstick", (i + 1) % n, d2);
          }
    }
}
```

Tuple Spaces provide:

- Nondeterminism
- Structured naming similar to "select" in relational DB, and pattern matching in AI.
- Time uncoupling (Communication between time-disjoint processes)
- Distributed sharing (Variables shared between disjoint processes)

Implementazione

- Richiesta efficienza nel pattern matching all'interno dello spazio delle tuple
- Preprocessing
  - A: out(“x”, i); B: in(“x”, ?i)
  - x è costante durante la compilazione, quindi si mettono A e B in contatto diretto (variabile condivisa o scambio di messaggi).
  - Problemi: con compilazioni separate o linguaggi interpretati.
- Restringere la ricerca su una chiave singola
  - Se le anti tuple hanno una sola variabile al loro interno, è possibile adottare l'hashing per la memorizzazione ed il recupero delle tuple , ottenendo l'accesso ad una tupla in tempo costante

Spazi delle tuple multipli

- lo spazio delle tuple non è centralizzato , la quantità di tuple da controllare diminuisce aumentando l'efficienza.