Architetture Software

organizzare la struttura

Andrea Polini

Ingegneria del Software
Corso di Laurea in Informatica

Scopo della fase di design

Nella fase di design devono essere prese decisioni sulla struttura del software. Questo implica la scomposizione del software in elementi più semplici e lo studio della loro composizione.

- controllo
- comunicazione

Molte scelte possibili....quale è quella più adatta e quali sono vantaggi/svantaggi di ogni scelta?

Scelte architetturali inevitabilmente influenzano specifica dei requisiti. In particolare con riferimento alla specifica dei sottosistemi

Ingegneria del Software fornisce supporto alle decisioni ma certamente la fase di design richiede una buona dose di creatività ed esperienza

Architettura software

Le decisioni di scomposizione del sistema in sottosistemi, la strutturazione dei dati, la definizione della struttura e dei "paradigmi" di comunicazione costituiscono quella che viene comunemente detta architettura software.

Lo studio delle architetture software costituisce una disciplina emergente ancora non completamente matura. Sistema identificato come assemblaggio di due tipi di elementi:

- componenti
- connettori

Un insieme di tali elementi viene composto in un sistema attraverso la definizione di una configurazione

Documentazione e scopi

Al solito la fase di specifica architetturale produrrà documentazione utile alla descrizione/compresione dell'architettura.

Scopi principali di tale documentazione sono:

- Comunicazione tra gli attori
- Analisi del sistema risultante
- Riuso

Architettura e QoS

Le scelte architetturali sono fortemente influenzate da fattori relativi a proprietà non funzionali.
Come è possibile agire:

- Performance: localizzare elementi critici all'interno di specifici componenti. Ridurre comunicazione tra questi.
- Security: strutturazione a strati dove gli strati più bassi forniscono supporto a quelli più alti
- Safety: ridurre le componenti critiche in modo da semplificare l'analisi
- Availability: semblificare la sostituzione di elementi del software senza dover fermare il sistema e includere meccanismi di replicazione
- Mantainability: ridurre la dimensione dei componenti

Al solito sarà necessario mediare!

Esperienza porta a definizione e riconoscimento di pattern architetturali che godono di particolari proprietà

Rappresentazione delle architetture

Uso di diagrammi schematici “scatole e linee” (componenti e controllo). È però evidente come l’informazione trasportata da un tale diagramma sia piuttosto vaga e la natura delle relazioni poco esplicita

Esempi di linguaggi formali di specifica architetturale (Architectural Desciption Languages - ADL)

- Darwin
- C2
- Kobra
- Wright
- . . .

Differenti linguaggi “semplificano” strutturazione in accordo a differenti stili architetturali.

Con riferimento a quanto appreso nel corso di laboratorio??

Rappresentazione delle architetture

Uso di diagrammi schematici “scatole e linee” (componenti e controllo). È però evidente come l’informazione trasportata da un tale diagramma sia piuttosto vaga e la natura delle relazioni poco esplicita

Esempi di linguaggi formali di specifica architetturale (Architectural Desciption Languages - ADL)

- Darwin
- C2
- Kobra
- Wright
- . . .

Differenti linguaggi “semplificano” strutturazione in accordo a differenti stili architetturali.
Con riferimento a quanto appreso nel corso di laboratorio??

Questioni Rilevanti

- C'è un architettura generica che si adatta alla richiesta e che può soddisfare i requisiti funzionali ed extra-funzionali?
- Il sistema dovrà essere distribuito su più machine?
- Ci sono stili architetturali che meglio si adattano al sistema?
- Approccio per strutturare il sistema?
- Come le varie componenti saranno decomposte in moduli?
- Quali strategie di controllo verranno utilizzate all'interno dei componenti?
- Quali valutazioni devono essere condotte?
- Come documentare la specifica architetturale?

Tipiche descrizioni architetturali

Un'architettura può essere definita cercando di modellare diversi aspetti del sistema:

- Modello strutturale
- Moello dinamico dei processi
- Modello delle interfacce
- Modello di relazione
- Modello di deployment

UML definisce diversi diagrammi che permettono di rappresentare tali informazioni

Stili Architetturali

Esperienza ha portato a riconoscere che certe organizzazioni dei sistemi godono di caratteristiche “favorevoli”.

- Layered (Modello Stratificato)
- Shared Memory
  - Repository - Blackboard
  - Rule Based
- Dataflow styles
  - Batch-Sequential
  - Pipe and filter
- Implicit invocation
  - Publish-Subscribe
  - Event Based
- Peer to peer

Layered - Modello stratificato

Sistema strutturato a livelli. Ogni livello fornisce un servizio più “astratto” ai livelli superiori.

- Stratificazione favorisce sviluppo incrementale ed evoluzione
- Modifiche ad uno strato risultano localizzate se non vengono modificate le interfacce. Altrimenti in generale propagazione al solo livello superiore.
- Non è sempre semplice definire struttura a strati per un sistema

Approccio con Macchine virtuali esempio di Layered architecture

Layered - Client-Server

- Funzionalità del sistema distribuite logicamente all'interno di un insieme di componenti serventi.
- Definizione di componenti clienti che sono quei componenti che hanno invece necessità di accedere ai servizi.
- Non necessariamente clienti e serventi si trovano su macchine differenti...

Client-Server Pro e Contro:

- Certamente paradigma particolarmente adatto in ambito distribuito (+).
- Disaccoppia fortemente parti del sistema (serventi da clienti) rendendo facile aggiungere componenti client e server (+)
- Modifiche ad un servente possono portare a revisioni importanti dei cliente (-) particolarmenti difficoltose in caso di distribuzione.

Shared Memory - Repository

- Si considerino sottosistemi che necessitano di comunicare pesantemente. Due possibili soluzioni:
  - Spazio condiviso
  - Invio dati mantenuti localmente
- Tipicamente la soluzione è quella di strutturare il sistema attorno ad un repository comune tra tutti i componenti del sistema (ogni componente può poi essere basato su un repository interno).
- Interazione tra i componenti avviene accedendo e modificando i dati nel repository.
- Tipica soluzione in strumenti CAD and CASE

Shared Memory - Repository

pro e contro

- Metodo semplice per scambiare grosse moli di dati tra i componenti (+)
- Non di meno ci deve essere un accordo sul formato dei dati. Difficile riuso (-) Possibili problemi di performance (-)
- Un sottosistema non si interessa di come gli altri sistemi utilizzeranno i dati. Disaccoppiamento (+) Modifiche al formato possono diventare complesse (-)
- Centralizzazione dei dati può creare problemi di affidabilità.
- Attività di gestione dei dati semplificata (+) ma poco flessibile e difficile evoluzione (-)
- Facile integrare nuovi componenti se questi sono “a conoscenza” del formato
- Distribuzione dei dati per migliorare performance complica gestione

Repository di tipo blackboard

Shared Memory - Rule Based

In questo stile esistono i componenti che possono produrre fatti e richiedere lo stato della *knowledge base* che costituisce lo spazio di informazioni condivise.
I componenti che costituisco l'architettura sono

- User Interface
- Inference Engine: è un componente capace di ragionare su regole della forma "if . . . then . . . "
- Knowledge Base

Caratteristiche:

- Difficile predirre il comportamento del sistema e testarlo
+ Facile da modificare
+ Facile realizzare prototipi esplorativi

Shared Memory - Rule Based

In questo stile esistono i componenti che possono produrre fatti e richiedere lo stato della *knowledge base* che costituisce lo spazio di informazioni condivise.
I componenti che costituisco l'architettura sono

- User Interface
- Inference Engine: è un componente capace di ragionare su regole della forma "if . . . then . . . "
- Knowledge Base

Caratteristiche:

- \- Difficile predirre il comportamento del sistema e testarlo
- \+ Facile da modificare
- \+ Facile realizzare prototipi esplorativi

Data-flow styles - Batch Sequential

Lo stile è usato quando complesse computazioni devono essere svolte su grosse moli di dati. I componenti coinvolti sono:

- Sorgente dati: definisce ed ordina i dati sui quali deve essere svolta la computazione
- Scheduler: organizza le richiesta per il processamento
- Componente computazionale: implementa gli aspetti computazionali relativi al problema che esegue su tutto il dataset fornito producendo un “blob” di dati risultanti. La computazione avviene in maniera asincrona

Caratteristiche:

+ Possibile ottimizzare uso di risorse costose
- Difficile interazione
- Complessi cicli di debugging

Data-flow styles - Batch Sequential

Lo stile è usato quando complesse computazioni devono essere svolte su grosse moli di dati. I componenti coinvolti sono:

- Sorgente dati: definisce ed ordina i dati sui quali deve essere svolta la computazione
- Scheduler: organizza le richiesta per il processamento
- Componente computazionale: implementa gli aspetti computazionali relativi al problema che esegue su tutto il dataset fornito producendo un “blob” di dati risultanti. La computazione avviene in maniera asincrona

Caratteristiche:

+ Possibile ottimizzare uso di risorse costose
- Difficile interazione
- Complessi cicli di debugging

Data-flow styles - Pipe and Filter

In questo stile viene usato quando si richiede computazione complessa che viene organizzata su diversi stadi che scambiano dati I componenti sono:

- Stadi del pipeline: sono componenti che trasformano il flusso di dati in ingresso in un flusso di uscita da fornire allo stadio più a valle

Caratteristiche:

+ Facile aggiungere stadi al pipeline con effetti locali
+ Modo efficace per parallelizzare attività
- Necessità di definire e gestire molti formati di interscambio
- Ridotta interattività
- Difficile organizzare sistemi generali in accordo allo stile

Tipicamente stile usato in sistemi di processamento di linguaggi

Data-flow styles - Pipe and Filter

In questo stile viene usato quando si richiede computazione complessa che viene organizzata su diversi stadi che scambiano dati I componenti sono:

- Stadi del pipeline: sono componenti che trasformano il flusso di dati in ingresso in un flusso di uscita da fornire allo stadio più a valle

Caratteristiche:

+ Facile aggiungere stadi al pipeline con effetti locali
+ Modo efficace per parallelizzare attività
- Necessità di definire e gestire molti formati di interscambio
- Ridotta interattività
- Difficile organizzare sistemi generali in accordo allo stile

Tipicamente stile usato in sistemi di processamento di linguaggi

Implicit Invocation - Publish Subscribe

Lo stile è utilizzato per permettere di disaccoppiare i componenti che interagiscono che utlizzeranno il meccanismo per pubblicare e/o ricevere informazioni e dunque effettuare computazione. I componenti inclusi nello stile sono:

- Publishers: sono componenti che tipicamente producono informazioni che possone essere di rilievo per altri componenti
- Subscribers: sono componenti interessati a ricevere informazioni
- Componente di distribuzione (middleware): permette di accoppiare i publisher ed i subscribers

Caratteristiche:

+ Alto disaccoppiamento dei componenti (facile aggiungere e/o rimuovere componenti)
- Generalmente non adatto a contesti real time
- Difficile predirre comportamento di sistema complesso

Implicit Invocation - Publish Subscribe

Lo stile è utilizzato per permettere di disaccoppiare i componenti che interagiscono che utlizzeranno il meccanismo per pubblicare e/o ricevere informazioni e dunque effettuare computazione. I componenti inclusi nello stile sono:

- Publishers: sono componenti che tipicamente producono informazioni che possone essere di rilievo per altri componenti
- Subscribers: sono componenti interessati a ricevere informazioni
- Componente di distribuzione (middleware): permette di accoppiare i publisher ed i subscribers

Caratteristiche:

+ Alto disaccoppiamento dei componenti (facile aggiungere e/o rimuovere componenti)
- Generalmente non adatto a contesti real time
- Difficile predirre comportamento di sistema complesso

Implicit invocation - Event Based

Simile a stile precedente dove però eventi sono distribuiti a tutti componenti che decidono poi indipendentemente di reagire o meno all'evento. Basato su utilizzo di event-bus.

Peer to Peer

In questo caso il sistema viene strutturato come una rete di nodi computazionali autonomi ed omogenei che possono gestire dati differenti (dati e controllo decentralizzati). Componenti:

- Peers: nodi omogenei che interagiscono tra loro, continuamente ponendo richieste e rispondendo a richieste degli altri peer (ogni nodo può agire da client e da server)
- Server: in alcuni casi sono previsti nodi con compiti specifici in particolare in relazione a servizi di lookup di informazioni.

Caratteristiche:

+ struttura che permette di ottenere affidabilità e disponibilità del servizio
+ facile ottenere scalabilità del sistema
- Generalmente limitato a specifici contesti applicativi per la condivisione di risorse distribuite