5. 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 composti 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

Organizzazione dei sistemi software

stili architetturali

Esempi di paradigmi per la definizione della struttura di un sistema:

- Modello del “repository”
- Modello Cliente-Servente
- Modello a strati

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

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

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.

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

Decomposizione modulare

Sottosistemi vengono strutturati in moduli. Distinzione tra modulo e sottosistema?

Due approcci fondamentali alla decomposizione in moduli:

- decomposizione orientata agli oggetti - il sottosistema viene strutturato come un insieme di oggetti interagenti raggruppati in package.
- decomposizione orientata alle funzioni (pipelining) - le attività del sottosistema vengono strutturate secondo un insieme di funzionalità da eseguire in sequenza.

Decomposizione object-oriented

Oggetti e definizione di precise interfacce.
Classici vantaggi del paradigma OO

- localizzazione delle modifiche (incapsulamento ed interfacce stabili)
- mapping su concetti reali semplice
- possibilità di riuso

Problemi sono principalmente legati alla necessità di riferire esplicitamente gli oggetti utilizzati. Modifiche ad interfacce diventano difficilmente gestibili.

Pipelining

Il sottosistema viene strutturato come un insieme di attività da eseguire in sequenza.
Tipicamente adatta a computazione che implica successive manipolazioni di un flusso di dati
I vantaggi principali riguardano:

- funzioni possono essere facilmente riutilizzate
- corrisponde spesso a come le persone strutturano o pensano alle loro attività
- aggiungere una funzionalità diventa piuttosto semplice
- possibilità di avere concorrenza può essere semplificata

Gli svantaggi rigurdano la difficoltà di implementare sistemi interattivi, e la necessità di stabilire un formato di interazione tra i diversi elementi del pipeline.

Stili di Controllo

Il controllo si riferisce a come il flusso del controllo si propaga tra i vari sottosistemi di un sistema software.
Principalmente si può optare tra due stili principali:

- Controllo centralizzato - esiste un sottosistema che controlla gli altri ed ha anche il compito di far partire ed arrestare il sistema.
- Controllo basato su eventi - ogni sottosistema è capace di rispondere indipendentemente ad eventi esterni.

Controllo centralizzato

Si distingue tipicamente tra due tipologie fondamentali:

- modelli call-return - il sistema parte con una routine principale (main) che passa il controllo alle altre routine le quali potranno fare lo stesso. Ogni routine si aspetta comunque di riavere il controllo “indietro” e dovrà dunque restituirlo. (No concorrenza)
- modelli con gestore - esiste un sottosistema (gestore) che può far partire concorrentemente più attività che possono procedere in parallelo.

Controllo centralizzato

Struttura del controllo è piuttosto semplice e semplifica la fase di analisi

Diventa difficile inserire e rispondere a comportamenti eccezionali al sistema che si possono verificare ad esempio come conseguenza di eventi esterni.

Controllo basato su eventi

L'andamento del sistema e del "Processo" è guidato da eventi generati esternamente (Sistemi guidati da eventi).

Certamente particolarmente adatto nel caso di sistemi interattivi, dove gli eventi esterni "scatenano" l'esecuzione di particolari procedure.

Tipicamente diversi modelli possono essere pensati per gestiti gli eventi:

- Modelli Broadcast e Publish/Subscribe
- Modelli ad Interrupt