Lorenzo Cioni

Il modello data-flow come design tool per lo sviluppo di applicazioni per il Digital Signal Processing

(XXIV Convegno dell'Associazione Italiana di Acustica, Trento12-14 Giugno 1996)

SOMMARIO

Scopo del presente articolo è quello di illustrare un approccio alla progettazione di software applicativo orientato al Digital Signal Processing e ispirato al modello data-flow. Viene proposta una modalità di progettazione di applicazioni guidata dal flusso dei dati fra i moduli che costituiscono gli elementi funzionali dell'applicazione in corso di sviluppo. Tale modalità fa uso dei grafi di flusso, tipici del modello data-flow, come strumenti di progettazione ed analisi. L'approccio proposto si presta a caratterizzazioni bottom-up, top-down o middle-out, in funzione della granularità dei moduli da cui il progettista parte per la specifica delle caratteristiche dell'applicazione.

INTRODUZIONE

Il presente articolo illustra alcune tecniche utilizzabili per la progettazione e la stesura di software applicativo orientato al Digital Signal Processing (DSP) ma la cui applicabilità è di tipo generale. L'idea di partenza ([Cio95]) si fonda sulla considerazione empirica che quanto più una applicazione è semplice tanto più facilmente può essere progettata, implementata e manutenuta. Tale considerazione è valida nell'ambito di grosse organizzazioni dedicate allo sviluppo di software applicativo e/o di sistema ma mostra tutta la sua potenza ogni volta che l'utente finale coincide con il progettista/sviluppatore di software applicativo o appartiene a tale comunità. Il requisito della semplicità trova "conforto teorico" nei concetti di modulo e modularità ([De73a], [De73b]) e nel paradigma dell'Object Oriented Programming, con le sue tecniche per lo sviluppo di software applicativo, ma si scontra con la necessità pratica di disporre di applicazioni in grado di risolvere problemi di complessità crescente. Scopo dei paragrafi che seguono è quello di mostrare come l'analisi del flusso dei dati fra i moduli componenti una applicazione, consentendo l'adozione di un approccio di tipo data-flow, possa essere un utile strumento per la progettazione di software applicativo che sia potente, versatile ed affidabile. Si sottolinea il fatto che, a questo livello di astrazione, i singoli moduli tendono ad essere più applicazioni elementari che operazioni elementari, nel senso che l'applicazione del modello viene reiterata fino alla definizione di moduli di granularità non minima (vedi figura 2).

IL DOMINIO DELLE APPLICAZIONI: IL DSP

Gli elementi fondamentali di un qualunque sistema per il DSP sono essenzialmente riconducibili ai seguenti ([NaIns94], [Cio94], figura 1): acquisizione dei dati, memorizzazione dei dati, analisi dei dati e presentazione dei dati. Già da questa analisi preliminare risulta evidente la convenienza di un approccio modulare ai problemi combinato all'analisi del flusso dei dati fra i moduli, schematizzato dagli archi orientati.
I dati di partenza sono costituiti da porzioni di parlato, acquisite con svariate tecniche e memorizzate essenzialmente sotto forma di file, oppure da risultati di operazioni di analisi o di editing.

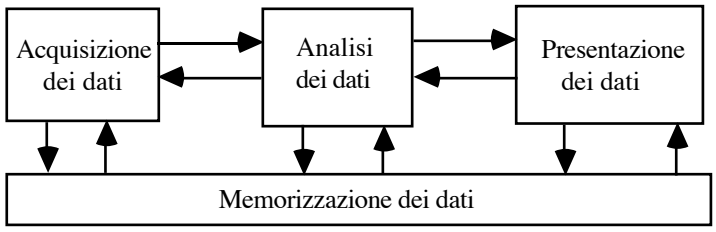

Figura 1

L'acquisizione dei dati ([NaIns94]) si basa su un insieme di tecniche oramai consolidate, fra le quali ci sembrano degne di nota sia quelle riconducibili all'uso dei protocolli IEEE488/GPIB e RS-232 sia quelle che fanno uso di schede per acquisizione di tipo plug-in o di schede interfacciate sul bus VXI/VME. L'analisi dei dati mira essenzialmente sia a trasformare dati

grezzi in informazioni significative sia a ottenere nuovi dati su diversi domini di rappresentazione, mediante tecniche di elaborazione nel dominio del tempo o della frequenza. La memorizzazione dei dati permette il salvataggio dei dati prodotti sia dall'acquisizione sia dall'analisi in strutture permanenti quali file o data-base ([Cio94]). La presentazione dei dati, infine, prevede essenzialmente l'uso di tecniche di visualizzazione e post-processing (editing interattivo e non) dei dati.

IL FLUSSO DEI DATI COME STRUMENTO DI PROGETTAZIONE

In molti casi, quando ci si appresta a scrivere una applicazione (in particolare per il DSP) l'insieme delle operazioni che si vuole rendere disponibili attraverso di essa è ben noto ed altrettanto noti sono i domini di rappresentazione su cui l'applicazione deve lavorare. A questo punto, tuttavia, spesso non è chiaro quale sia il modo più semplice di decomporre tale applicazione in moduli sempre più semplici (top-down) oppure se questa possa essere ottenuta componendo moduli già disponibili o di facile implementazione (software reuse e bottom-up) oppure, infine, se non sia preferibile impostare la progettazione partendo dalla definizione di moduli di media complessità che sia possibile comporre in moduli più complessi ma che vadano, a loro volta, implementati (middle out).
Un modo di gestire la complessità della progettazione di software applicativo è rappresentato dall'esame del flusso dei dati fra i moduli che vengono via via individuati nel corso dell'analisi o che sono utilizzati nel corso della progettazione. Il tool proposto è un tool di tipo descrittivo, che non preclude nessuna delle tecniche implementative oggi in uso ma che spinge verso la definizione di applicazioni semplici e facilmente componibili, che possono co-operare per lo svolgimento di compiti complessi. L'analisi dei flussi di dati, inoltre, incoraggia il riutilizzo di software e sia l'integrazione dei sistemi di calcolo sia lo sfruttamento di ambienti distribuiti ed eterogenei (ad esempio file/disk server, I/O server e architetture particolari dedicate ad elaborazioni CPU intensive).
La figura 1 può essere vista, quindi, come il primo passo di raffinamento del processo di progettazione di una applicazione general-purpose per il DSP secondo un approccio top-down: a questo livello ciò che interessa mettere in evidenza sono i flussi dei dati fra i moduli piuttosto che i dettagli algoritmici di implementazione di una particolare operazione su certi dati in un dato dominio di rappresentazione.

IL MODELLO DATA-FLOW BASE E ALCUNE SUE ESTENSIONI

Il modello data-flow ([BaToVa87], [Tre87]) vede una computazione come descrivibile in generale mediante una rete data-flow, nei casi più semplici mediante un grafo, in molti casi aciclico. Dato un grafo o una rete, i nodi rappresentano i moduli funzionali mentre gli archi rappresentano sia i canali attraverso cui i moduli si scambiano i dati sia le dipendenze fra i diversi moduli. I flussi dei dati sono rappresentato dal transito di token lungo gli archi e l'evoluzione di una computazione è rappresentata dalla successione dei transiti dei token sugli archi. I singoli moduli sono caratterizzati da un legame di tipo funzionale fra i dati in ingresso e quelli in uscita e possono aversi nodi di input con soli archi in uscita e di output con soli archi in ingresso. Nel caso del modello data-flow la computazione è governata dalla disponibilità dei dati in ingresso ai vari moduli (data driven). Per sua natura, la computazione è, inoltre, completamente asincrona ed è caratterizzata da un elevato parallelismo fra i moduli, il più alto ottenibile, date le dipendenze logiche fra i dati coinvolti nella computazione stessa. Il fatto che la computazione sia asincrona richiede, a rigore, la presenza di code di token di dati fra i nodi della rete o del grafo che rappresentano la struttura della computazione. Un altro requisito è che una computazione sia deterministica, ossia che il risultato sia indipendente dall'ordine di valutazione dei token da parte dei nodi: tale requisito si traduce nel modello nell'introduzione di vincoli più o meno stretti per la caratterizzazione dei nodi. Dal momento che il modello data-flow viene proposto come tool descrittivo, le caratteristiche utili ai nostri scopi sono riconducibili alla natura asincrona e ad elevato parallelismo delle computazioni ed al fatto che queste sono descrivibili mediante uno strumento semplice e potente quale sono i grafi e le reti data-flow. Per gli scopi cui il modello è dedicato, sono utili alcune estensioni al modello data-flow (che per motivi di spazio ci si limita ad elencare) quali la possibilità di avere oggetti passivi nei grafi, ad esempio i file di configurazione etichettati come (#) in figura 2, e la possibilità di definire moduli che oltre ad avere archi in ingresso e/o in uscita sono caratterizzati da interazioni asincrone con l'utente (vedi modulo etichettato come Presentazione in figura 2): si ha la possibilità di introdurre moduli la cui evoluzione è

governata dall'esterno oltre che dalla disponibilità di dati sugli archi in ingresso (modalità di valutazione stretta e lasca).

ESEMPI DI APPLICAZIONE

La figura 2 illustra brevemente alcune ulteriori possibili applicazioni alla progettazione di software per il DSP dell'approccio proposto nei paragrafi precedenti. L'esempio non è esaustivo ma illustra come l'attenzione ai flussi dei dati possa recare benefici alla progettazione. Nella figura, gli archi orientati individuano i flussi dei dati fra i moduli e non sono in alcun modo in relazione con il passaggio del controllo fra i moduli stessi. Leggendo la figura dall'alto in basso si assume un punto di vista top-down, dal basso in alto, data la granularità dei moduli, si assume un punto di vista middle-out oppure un punto di vista bottom-up in dipendenza del fatto che i moduli interessati sono primitivi per il nostro dominio oppure no: i moduli (C) e (B) non sono in genere primitivi mentre di sicuro lo è il modulo (D) (uno switch settabile dall'utente) e possono esserlo i moduli etichettati come EPD e F0.
Nella figura 2 l'applicazione (A), che da un punto di vista utente è un modulo unico, con una sua interfaccia e un suo proprio set di comandi, nasconde in realtà un insieme di moduli: uno che effettua l'analisi e la presentazione dei dati, uno che gestisce la navigazione all'interno di un file system o di un data base, uno per la memorizzazione di nuovi file risultato di operazioni di analisi o di editing (interattivo e non) ed altri due per la gestione dell'acquisizione e dell'ascolto. I moduli (B) e (C) costituiscono, globalmente, una applicazione (utilizzabile autonomamente) in grado di eseguire I/O in un ambiente distribuito con server locali o remoti. Entrambi i moduli sono infatti decomponibili a loro volta in un modulo client e in più moduli server: il client fornisce un'interfaccia con l'utente o con una applicazione che necessita di effettuare, rispettivamente, acquisizione o ascolto di speech e può interagire con più moduli server, locali o remoti, che svolgono il compito effettivo di colloquiare con l'hardware di I/O: in figura si vedono due server, uno locale di bassa qualità (8kHz e 8bit/campione) ed uno remoto di qualità decisamente migliore (8/10/12/16kHz e 16bit/campione). Analogamente, l'applicazione che si occupa di Analisi&Presentazione può essere pensata come composta da due moduli, uno che si occupa di visualizzare i dati (Presentazione) ed uno che gestisce le operazioni di analisi (Analisi). Di nuovo, il modulo che gestisce la presentazione può essere decomposto in una parte che interagisce con il window manager, una che si occupa di gestire le rappresentazioni interne ed una che svolge le operazioni di editing interattivo, di segmentazione e di labelling.
Il modulo che gestisce le operazioni di analisi, a sua volta, è caratterizzabile sia nel dominio del tempo sia nel dominio della frequenza ed è possibile inserire nel grafo un modulo che ha il compito di eseguire le operazioni di editing non interattivo. La decomposizione illustrata in figura è orientata alla progettazione di un modulo per l'estrazione della F0 mediante Sift. A questo punto non v'è chi non vede come la decomposizione sia di tipo funzionale mentre il legame fra i moduli è associato ai passaggi dei dati fra i moduli che compongono una applicazione.

CONCLUSIONI

Semplicità ed economia di pensiero sono o piuttosto dovrebbero essere i principi alla base della progettazione di software applicativo (e di sistema). L'approccio proposto spinge alla definizione di applicazioni complesse mediante la definizione di moduli il più possibile "utili di per se", composti fra loro sulla base dell'analisi dei flussi dei dati. Il modello risultante, sebbene di tipo astratto, si presta ad una agevole implementazione anche su architetture distribuite, multitasking o multiprocessore, qualora i singoli moduli siano fatti corrispondere a processi o gruppi di processi e si sfruttino ambienti di integrazione "nativi".

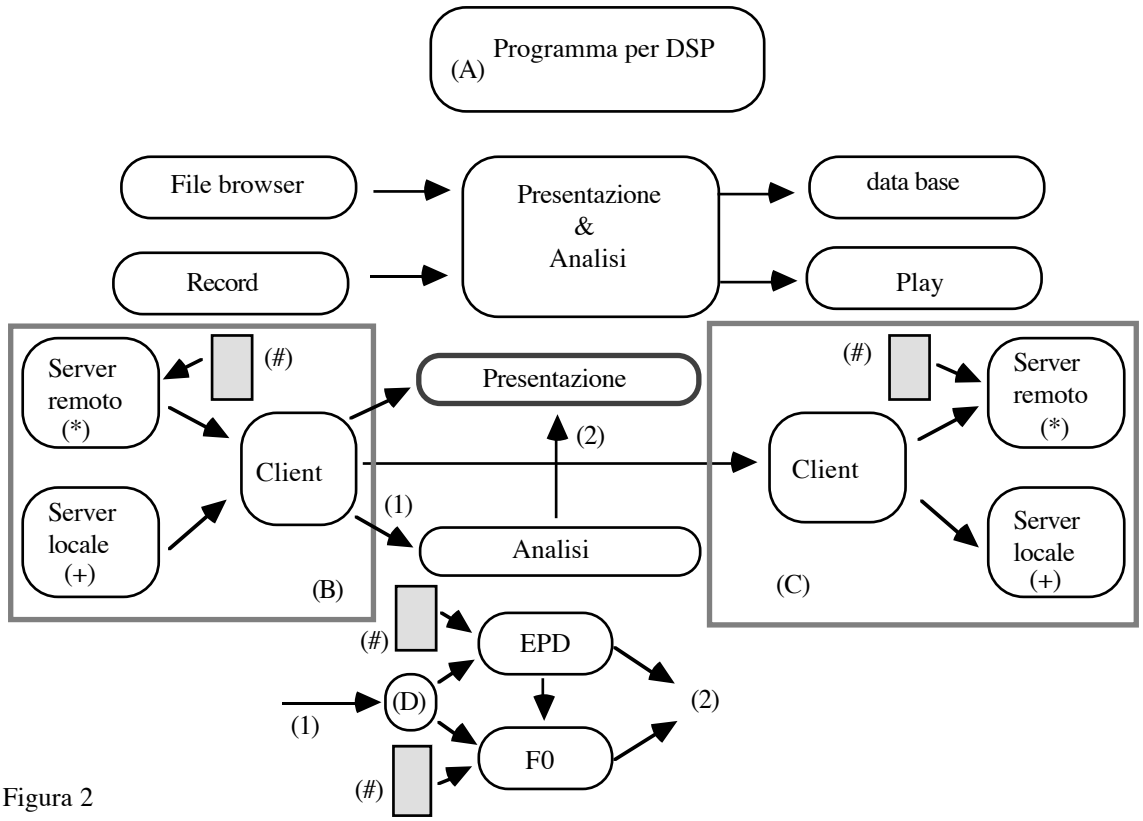

Figura 2

BIBLIOGRAFIA

[BaToVa87] F. Baiardi, A. Tomasi , M. Vanneschi, Architettura dei Sistemi di Elaborazione, Vol. 1, Franco Angeli, Milano, 1987.

[Cio94] L. Cioni, "A data-base for speech signal processing", atti dell'SST-94, 6-8 Dicembre 1994, Perth, Western Australia, Australia.

[Cio95] L. Cioni, "Co-operative principles in application design", 4th International Workshop on Software Engineering and Artificial Intelligence for High Energy Physics, Pisa, Italy, 3-8 April 1995.

[De73a] J. B. Dennis, "The design and construction of software systems", *Software Engineering. An advanced course*, eds. G. Goos and J. Hartmanis, Lecture Notes in Compter Science, Springer Verlag, Munich, 1973.

[De73b] J. B. Dennis, "Modularity", *Software Engineering. An advanced course,* eds. G. Goos and J. Hartmanis, Lecture Notes in Compter Science, Springer Verlag, Munich, 1973.

[NaIns94] National Instruments, *IEEE 488 and VXIbus Control, data Acquisition and Analysis*, Products Cathalog, 1994.

[Tre87] P.C. Treleaven, "The data-flow approach for MIMD multiprocessor systems", *Parallel Processing Systems. An advanced course*, ed. D. J. Evans, University Press, Cambridge, 1987.