L'insegnamento di Nuove Tecnologie di Programmazione: Alcune Esperienze e Considerazioni

Jean-Pierre BRIOT

LITP
Institut Blaise Pascal
4 place Jussieu, 75252 Paris Cedex 05, France
tel: +33 (1) 44 27 40 55
fax: +33 (1) 44 27 62 86
e-mail: briot@litp.ibp.fr

Abstract

Questo articolo discute dell'introduzione di nuove metodologie di programmazione nel campo dell'informatica. La nostra discussione parte da un'osservazione di un considerevole aumento di modelli metaforici di programmazione, estrapolati da varie discipline scientifiche. Un concetto base che si estrae da questi vari modelli è quello delle entità interattive ed attive. Tali entità, chiamate oggetti, sono caratterizzati da un comportamento concorrente, cioè eseguono i loro compiti ed interagiscono tra loro parallelamente. Questa nuova, emergente metodologia di programmazione è conosciuta con il nome programmazione concorrente orientata agli oggetti (in breve OOCP da object-oriented concurrent programming). Inizialmente introdurremo i concetti base della metodologia OOCP; in seguito analizzeremo le conseguenze nel campo didattico. Per meglio comprendere le caratteristiche della OOCP descriveremo brevemente una piattaforma software che è stata realizzata per poter studiare le conseguenze di questa nuova metodologia. Concluderemo con una discussione sulle esperienze ottenute nel campo pedagogico.

1 Nuovi Orizzonti nei Paradigmi di Programmazione

L'uso dei computers è ormai massicio nella nostra società. Inizialmente il loro uso era limitato nei settori scientifici o in quelle realtà industriali che richiedevano una particolare gestione di dati. Oramai non possiamo identificare settori che siano estranei all'utilizzo del computer. Questa reale invasione ha accellerato l'evoluzione di modelli di programmazione. Nella sezione successiva prenderemo in esame una delle più promettenti metodologie di programmazione particolarmente adatta alla futura generazione di computers e alle più innovative applicazioni.

1.1 Programmazione Metaforica

Il binomio programmazione-computer sta vivendo un momento di profonda trasformazione. Possiamo notare che i modelli di programmazione stanno migrando da modelli basati su fondamenti matematici o da modelli basati su sequenze di istruzioni verso modelli *metaforici*. I modelli metaforici identificano quei nuovi modelli di programmazione nati dall'esigenza di adottare modelli di computazione alternativi da utilizzare in specifiche aree scientifiche. Un esempio chiaro è fornito in letteratura dagli studi compiuti sulle reti (artificiali) neuroniche [PDP 86], intese come astrazione di concetti legati alla neurofisiologia. Numerosi altri esempi possono essere travati nella chimica (vedi i modelli computazionali usati per simulare la reazione/diffusione, ovvero la macchina astratta chimica [Berry and Boudol 90]), la genetica (algoritmi genetici [Goldberg 89]), l'etologia (simulazione di vespai), economia (modelli di marketing), ed altri tipi di sistemi auto-organizzativi [ECO 88]. Una forte tendenza riscontrata nei nuovi modelli di programmazione è la nozione di agenti cooperativi e inter-comunicanti, tendenza denominata Intelligenza Artificiale Distribuita (brevemente conosciuta con la sigla DAI), che eredita forti componenti dalla sociologia e dalla teorie organizzative [DAI 88].

1.2 Oggetti

La maggioranza di questi nuovi modelli di computazione propone la nozione di entità cooperative ed attive. Tale nozione è chiamata *pro-*

grammazione orientata ad oggetti (OOP in breve). Il suo modello computazionale è talmente semplice e generale da poter essere applicato in vaste aree di applicazione. Fondamentalmente si hanno due concetti base : *oggetti* che identificano la conoscenza (le informazioni ed i servizi che essi offrono), ed un protocollo di comunicazione/attivazione di oggetti chiamato *trasmissione di messaggi*. Naturalmente intervengono altri concetti molto importanti come *l'astrazione* e *l'ereditarietà*, che permettono di classificare e reutilizzare la descrizione di oggetti. Una delle prime esperienze effettuate su tali concetti risalgono agli anni '60 con il linguaggio Simula, ma dobbiamo aspettare il linguaggio Smalltalk [Goldberg and Robson 83]) per riscontrare una larga diffusione di questi nuovi concetti nella comunità scientifica. Uno dei più popolari calcolatori, il Macintosh della Apple, realizzato sulla base di questi concetti, ha cambiato la mentalità tradizionale nell'uso di un personal computer. Attualmente i meriti di questa nuova metodologia sono stati ampiamente evidenziati dall'Ingegneria del Software, in termini di vantaggi che si osservano in tutte le varie fasi che distinguono lo sviluppo software [Meyer 88]. Sfortunatamente, la presenza di un unico processore, rende la maggior parte dei linguaggi OOP puramente sequenziali.

1.3 Concorrenza

Nonostante questa (temporanea) realtà di sequenzialità la *concorrenza* rimane la caratteristica di base per la futura generazione di calcolatori. Questa caratteristica non è certo una pura illusione, dato che esiste già ora la disponibilità di architetture multi-computers che permettono l'esecuzione parallela dei programmi, architetture che subiscono continui miglioramenti in termine di prestazioni. Altre motivazioni che possono riscontrarsi a favore dell'uso della concorrenza, risiede sulla distribuzione della conoscenza, come viene sottolineata dalle reti di computers, e dagli aspetti di forti interazioni multiple esistenti in grossi programmi interattivi, come quelli realizzati per il controllo dei processi, in cui l'input dei dati ed il controllo possono essere multipli e distribuiti.

La programmazione concorrente è indipendente dai dettagli delle architetture scelte. Per questo motivo si dovrebbe distinguere la concorrenza (simultaneità logica di attività) dal parallelismo (simultaneità fisiche). La sincronizzazione assicura il coordinamento e l'integrità delle varie attività, specificando le condizioni sull'ordine degli eventi [Andrews and Schneider 83].

1.4 Oggetti + Concorrenza = Attori

La programmazione concorrente orientata agli oggetti (più brevemente OOCP) nasce dal connubio della programmazione orientata agli oggetti con la programmazione concorrente [OOCP 87]. Per raggiungere un livello ottimale da tale integrazione, è utile considerare gli oggetti come un'unità di attivazione, ed associare l'attività di scambio dei messsaggi alla funzione di sincronizzazione. In tal modo ogni oggetto possiede la sua attività ed autonomamente gestisce le possibili interazioni multiple. Tale strategia provoca la concorrenza di attività ed interazioni tra oggetti. Questi oggetti, concorrenti ed interattivi, sono chiamati spesso *attori*.

L'usuale programmazione (sequenziale) orientata agli oggetti viene vista come una restrizione tecnologica della metafora generale della programmazione (concorrente) orientata agli oggetti. La OOCP costituisce una valida metodologia per decomporre grossi programmi in un insieme di moduli cooperanti, e permette la loro esecuzione su adatte architteture di computer che lavorano in parallelo. Riassumendo, possiamo evidenziare i seguenti vantaggi : alto livello di espressione, modularità, dinamicità ed apertura. Anche se la OOCP rappresenta attualmente un nuovo campo in continua espansione, si possono già riscontrare decisivi influssi sulle archittetture a più processori (come la J-Machine [Dally et al. 89]) e numerose applicazioni nei sistemi per l'analisi dei segnali [Barry 89], controllo dei processi, sistemi per l'automazione dell'ufficio, e perfino nell'animazione...

2 Insegnamento

In questa sezione saranno discussi i vantaggi apportati da questa nuova metodologia di programmazione agli studenti e a programmatori tradizionali. Tale analisi proviene da un'esperienza personale di insegnamento ottenuta da corsi e seminari centrati sull'uso della programmazione (concorrente) orientata agli oggetti. Tale esperienza è stata accumulata in ambito di Università, Politecnici, ed Istituti private. Per aiutare l'utilizzo di tale tecnica di programmazione, è stato usato un sistema software che ha permesso anche una chiarificazione dei vari linguaggi di programmazione, e la sperimentazione di metodologie comparative.

2.1 Programmazione Concettuale

La programmazione orientata agli oggetti risulta più concettuale rispetto la programmazione convenzionale. Essa permette di focalizzare l'attenzione tra agenti, astraendo il programmatore dai dettagli della loro rappresentazione ed implementazione. Questo nuovo stile di programmazione risulta più assimilabile non solo per dei programmatori convenzionali, ma anche per utilizzatori che non hanno ancora un'esperienza di programmazione. Questa opinione è convalidata da una diretta esperienza didattica riscontrata in diversi ambienti e situazioni.

2.2 Dagli Agenti agli Attori

La programmazione orientata agli oggetti è largamente diffusa nella comunità dei programmatori. Una questione molto delicata è come trasferire questa esperienza nella nuova comunità dei programmatori che usano una metodologia di programmazione concorrente basata sugli oggetti. Come già abbiamo fatto notare nella sezione la programmazione concorrente orientata agli oggetti è una generalizzazione della programmazione ad oggetti. Si consideri che però che per introdurre il concetto di concorrenza è necessario sviluppare alcuni aspetti tipici della programmazione ad oggetti. Questa situazione ha determinato una schiera numerosa di prototipi di linguaggi OOCP, dovuta anche alla recente ed intensa attività di ricerca su questo nuova area [OOCP 87]. Notiamo che non è sempre facile analizzare e confrontare questi linguaggi e delineare precisamente le loro caratteristiche implementative dato che un confronto semantico richiede un'opera di astrazione dai vincoli sintattici ed dalle scelte implementative. Si noti che la descrizione e la classificazione sono vantaggi apportati dalla programmazione ad oggetti, che hanno condotto all'implementazioni di piattaforme di sviluppo basati sulla OOP (si consideri per esempio Smalltalk-80 [Goldberg and Robson 83]) per la modellizzazione di linguaggi OOCP.

2.3 La Piattaforma Actalk

Questa piattaforma introduce gli attori nel mondo degli oggetti. Abbiamo scelto come linguaggio di programmazione ed ambiente di sviluppo Smalltalk-80 per i motivi discussi precedentemente. Come conseguesta di tale innesto abbiamo chiamato la nostra piattaforma *Actalk*, dall'unione di *Act*ors e Small*talk* [Briot 89]. Actalk fornisce un valido supporto per una descrizione chiara della programmazione concorrente ad oggetti ed è stata utilizzata come strumento sperimentale in numerosi corsi sulla OOCP. Inizialmente discuteremo delle scelte di progetto che hanno condotto alla realizzazione attuale del sistema. In seguito mostreremo rapidamente lo stato attuale del sistema ed concluderemo con delle riflessioni sui suoi scopi pedagogici. Una descrizione più approfondita di Actalk è data in [Briot 90].

2.3.1 Scelte Progettuali

Per progettare un sistema volendo integrare vari linguaggi OOCP in un unico ambiente di sviluppo, abbiamo osservato i seguenti obiettivi:

- *uniformità e modularità*
  Abbiamo unificato i vari linguaggi OOCP in un ambiente comune in modo da analizzarli e definirli in modo incrementale. Per raggiungere questo principio, abbiamo scelto il paradigma di programmazione ad oggetti. Gli attori sono stati introdotti in un ambiente sequenziale OOP, definendo un sotto-mondo di attori innestati in un mondo di oggetti, senza stravolgere il sistema soggiacente.

- *minimalità ed estendibilità*
  Volevamo esprimere la semantica più generale dei linguaggi OOCP partendo da un nucleo minimale per poterlo, in futuro, estenderlo. Abbiamo quindi definito una architettura minimale estendibile dalla proprietà di ereditarietà per simulare e classificare i vari modelli OOCP.

- *ambiente integrato*

  Non volevamo limitare il nostro sistema ad un semplice modello semantico o ad una grezza implementazione. Lo scopo era di fornire un ricco ambiente di programmazione di tipo OOCP. Per questo motivo, Smalltalk-80 risulta senza dubbio il sistema più flessibile ed adatto.

Scegliendo Smalltalk-80 otteniamo un'implementazione minimale ed omogenea, perchè tutte le entità richiedono la costruzione di attori: oggetti, classi, messaggi, e tutte le entità richiedono l'uso della concorrenza: processi, semafori, sono già disponibili in Smalltalk-80. In tal modo questa implementazione è sia ad alto livello che minimale. Ciò permette di focalizzarci sulla semantica della OOCP e sulle differenze dei tradizionali OOP, senza dover concentrarci su concetti di basso livello.

2.4 Pedagogia

Actalk è un ambiente integrato racchiuso in Smalltalk-80, usato come strumento di base per classificare e realizzare linguaggi OOCP. Dal perfetto connubio tra attori ed oggetti, Actalk è usato come base per studiare le relazioni e combinazioni tra oggetti ed attori. Actalk è stato usato con successo in molti corsi, seminari e tutorial per poter descrivere la transizione dalla programmazione ad oggetti verso la programmazione concorrente ad oggetti. Per la sua natura minimale di implementazione, Actalk permette di rendere immediatamente accessibile i concetti principali della OOCP. Actalk viene inoltre usato attualmente per descrivere i più diffusi modelli e linguaggi OOCP. Tali estensioni usano l'ereditarietà per poter modellare le semantiche degli OOCP e per poterle opportunamente compararle. Esempi di tali estensioni sono il modello computazionale definito da Gul Agha [Agha 86], il linguaggio di programmazione ABCL/1 realizzato da A. Yonezawa e dalla sua equipe [ABCL 89], ed il linguaggio POOL [America 87]. La simulazione di altri linguaggi concorrenti (o CSP) è stata condotta dall'equipe del Prof. Jean Bézivin dell'Università di Nantes. Inoltre Actalk è stato utilizzato per realizzare diversi prototipi multi-agenti dall'equipe del Prof. Jacques Ferber del laboratorio Laforia dell'Istituto Blaise Pascal a Paris. Uno di questi prototipi è il sistema multi-agente Mages [Bouron et al. 90]. Tali prototipi sono stati applicati a problemi combinatoriali fornendo interessanti risultati. Tutte queste esperienze ci convincono dei benefici che derivano dal nostro approccio basato sulla descrizione progressiva di concetti della OOCP, e di una facile esperimentazione di essa, grazie all'esistenza del sistema Actalk (e della sua base di realizzazione Smalltalk-80). Attualmente è in fase di realizzazione un libro sull'esperienza ottenuta dall'utilizzo della programmazione concorrente orientata agli oggetti.

Conclusione

Sono stati introdotti i concetti base della metodologia della Programmazione Concorrente Orientata agli Oggetti (OOCP), che stimolano la decomposizione dei programmi e permettono la loro esecuzione su architetture multi-computers. Abbiamo discusso su alcune conseguenze provocate da tale introduzione che sono state verificate su campioni di programmatori e non programmatori. In seguito abbiamo descritto una piattaforma di programmazione, chiamata Actalk, realizzata per verificare i concetti sperimentali della OOCP, sottolineando un suo utilizzo per scopi pedagogici. Il nostro obiettivo è di accumulare ulteriori esperienze nel processo di insegnamento e nella sperimentazione di questa nuova, emergente metodologia di programmazione.

Questo articolo è dedicata ad una "Gemella", chiamata Dominique. Grazie a Vincenzo Loia per avermi stimolato a riflettere sulle questioni derivanti dall'insegnamento di queste nuove metodologie di programmazione.

References

[ABCL 89] "ABCL: an Object-Oriented Concurrent System", edited by A. Yonezawa, Mit-Press, 1990.

[Agha 86] G. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems", Series in Artificial Intelligence, MIT Press, 1986.

[America 87] P. America, "POOL-T: A Parallel Object-Oriented Language", Pages 199-220, in [OOCP 87].

[Andrews and Schneider 83] G.R. Andrews and F.B. Schneider, "Concepts and Notations for Concurrent Programming", Computing Surveys, Vol. 15, No 1, March 1983.

[Barry 89] B. Barry, "Prototyping a Real-Time Embedded System in Smalltalk", Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications

(OOPSLA'89), Special Issue of Sigplan Notices, Vol. 22, No 12, pages 255-265, October 1989.

[Berry and Boudol 90] G. Berry and G. Boudol, "The Chemical Abstract Machine", ACM Conference on Principle of Programming Languages (POPL'90), San-Francisco CA, USA, 1990.

[Bouron et al. 90] T. Bouron, J. Ferber, and F. Samuel, "Mages: a Multi-Agent Testbed for Heterogeneous Agents", Proceedings of MAAMAW'90, Decentralized AI, Vol. II, North-Holland, 1991.

[Briot 89] J.-P. Briot, "Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment", Proceedings of the European Conference on Object-Oriented Programming (ECOOP'89), pages 109-129, Cambridge University Press, 1989.

[Briot 90] J.-P. Briot, "OOCP = OOP + C", Proceedings of the 3rd Conference on Object-Oriented Languages and Systems (Tools Pacific'90), pages 417-421, Sydney, Australia, November 1990.

[DAI 88] "Readings in Distributed Artificial Intelligence", edited by A. Bond and L. Gasser, Morgan Kaufman, 1988.

[Dally et al. 89] W.J. Dally et al., "The J-Machine: a Fine-Grain Concurrent Computer", Proceedings of Information Processing Congress (IFIP'89), pages 1147-1153, August 1989.

[ECO 88] "The Ecology of Computation", edited by B.A. Huberman, North-Holland, 1988.

[Goldberg and Robson 83] A. Goldberg and D. Robson, "Smalltalk-80: The Language and its Implementation", Series in Computer Science, Addison-Wesley, 1983.

[Goldberg 89] D.E. Goldberg, "Genetic Algorithms", Addison-Wesley, 1989.

[Meyer 88] B. Meyer, "Object-Oriented Software Construction", Series in Computer Science, Prentice-Hall, 1988.

[OOCP 87] "Object-Oriented Concurrent Programming", edited by A. Yonezawa and M. Tokoro, MIT Press, 1987.

[PDP 86] "Parallel Distributed Processing", edited by D.E. McClelland and J.L. Rumelhart, Vol. 1 and 2, MIT Press, 1986.