8. Bibliografia

[1] *Computer Aided Design. Modelling, System Engineering, CAD-Systems*, Lecture Notes in Computer Science, Springer Verlag (1980)

[2] G. Enderle, K. Kansy, G. Pfaff, *Computer Graphics Programming. GKS – The Graphics Standard*, Springer Verlag (1984)

[3] ACM Siggraph, *Status Report of the Graphic Standards Planning Committee of ACM /Siggraph Computer Graphics*, Siggraph–ACM 11, 3 (1977)

[4] *Programmer's Hierarchical Interactive Graphics Standard (PHIGS)*, ISO TC97/SCS/WG2/N209 (1985)

[5] F. M. Martins, J. N. Oliveira, *Graphics Programming with "Archetypes"*, Eurographics'85, North Holland (1985)

[6] P. Wisskirchen, *Towards Object-Oriented Graphics Standards*, Eurographics '85, North Holland (1985)

[7] H. Göttler, *Attributed Graph Grammars for Graphics*, in Graph-Grammars and their Applications to Computer Science, Lecture Notes in Computer Science, Springer Verlag (1982)

[8] P. J. W. ten Hagen, T. Hagen, P. Klint, H. Noot, H. J. Sint, A. H. Veen, *ILP. Intermediate Language for Pictures*, Mathematical Centre, Amsterdam (1980)

[9] W. F. Clocksin, C. S. Mellish, *Programming in Prolog*, Springer Verlag (1984)

[10] L. Naish, *Negation and Control in Prolog*, Lecture Notes in Computer Science, Springer Verlag (1986)

[11] J. Cohen, *Describing Prolog by its Interpretation and Compilation*, CACM 28, 12 (1985)

L'uso della storia in un Sistema per il Controllo di Accessi e Versioni

Alessandro Huber-Bachrich[1], Daniele Nardi[2]

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Buonarroti 12, I-00185 Roma, Italy

Sommario

Questo lavoro si inserisce nell'ambito delle problematiche riguardanti l'Ingegneria del Software, e considera la memorizzazione e l'utilizzo delle informazioni descriventi l'evoluzione di un progetto software. In particolare è stato preso in esame il Sistema per il Controllo di Accessi e Versioni (SCAV), e sono state realizzate funzioni per l'accesso e la manipolazione della storia di un progetto gestito con gli strumenti offerti da SCAV. Tale realizzazione si basa su una rappresentazione della storia in un formalismo chiamato calcolo degli eventi, che consente di descrivere l'evoluzione del sistema come sequenza di operazioni di SCAV. Il lavoro è stato realizzato in PROLOG, in cui si può direttamente esprimere il calcolo degli eventi.

1 INTRODUZIONE

Il mantenimento e l'analisi delle informazioni relative alla storia riveste una notevole importanza per una larga classe di sistemi, in particolare informatici, che si possano significativamente caratterizzare per mezzo di una nozione di stato e di transizioni dello stesso, causate dal funzionamento del sistema. In questo lavoro viene affrontato il problema della memorizzazione e dell'uso delle informazioni riguardanti l'evoluzione di un progetto software.

Prima di entrare nei dettagli, esemplifichiamo il concetto di storia riferendoci alle procedure di archiviazione per un sistema di calcolo. Il tipo di modellazione del sistema di calcolo che interessa in questo caso, è costituito semplicemente dallo stato della memoria di massa. L'attivazione delle procedure di archiviazione provoca una transizione dello stato, in quanto definisce una nuova situazione in termini di ciò che è accessibile agli

[1] Attualmente presso Tecsiel, via Sierra Nevada 110, Roma

[2] Ricerca parzialmente finanziata dall'Enidata

utenti in memoria di massa primaria e secondaria. Ovviamente questo modello è semplificato in modo da tenere conto solo degli aspetti rilevanti al problema della archiviazione. La politica con cui sono definite le procedure di archiviazione deriva dalla rilevanza (o utilità) di una informazione memorizzata, che è definita in base alle date ed alle frequenze di accesso. I dati non utilizzati vengono progressivamente memorizzati su supporti più compatti ed economici, ad accesso progressivamente più lento. Ad un certo punto le informazioni vengono considerate obsolete ed, eventualmente, vengono del tutto rimosse per fermare la crescita degli archivi.

In questa modellazione l'archivio rappresenta la storia del sistema, definita con un criterio che consente un tempo di accesso inversamente proporzionale alla frequenza di accesso. Quando si utilizzano delle descrizioni più significative rispetto al modello fornito dalla struttura a files, la storia può essere trattata in modo più accurato. Nel caso di un progetto software in corso di sviluppo e successivamente in fase di mantenimento, è stata presa in esame la possibilità di una caratterizzazione più precisa del progetto, fornendo ad esempio strumenti per la gestione e la memorizzazione efficiente delle versioni [4,10,14,15,16].

Il nostro lavoro ha come obiettivo quello di considerare la modellazione di un progetto software definita dal sistema Sistema per il Controllo di Accessi e Versioni (SCAV) ed il relativo concetto di storia [7,8]. Su tale base sono stati affrontati i seguenti problemi:

- l'individuazione delle informazioni rilevanti allo stato corrente del progetto;
- l'analisi delle informazioni contenute nella storia per l'organizzazione e gestione del progetto stesso.

La storia del sistema viene pertanto descritta mediante una base di conoscenze espressa sotto forma di calcolo degli eventi [12]. Una serie di funzionalità che utilizzano tale base di conoscenze sono state implementate in C-Prolog sul sistema SUN del Dipartimento di Informatica e Sistemistica dell' Università di Roma "La Sapienza".

2 LA STORIA IN SCAV

Il sistema SCAV [6,7,8] é stato progettato con l'obiettivo di offrire uno strumento di supporto alla cosiddetta "programmazione in grande". Esso infatti consente il controllo e l'organizzazione di un progetto software al responsabile del suo sviluppo e mantenimento, con particolare riguardo ai problemi del controllo degli accessi dei partecipanti al progetto e della gestione delle versioni rilasciate in fase di sviluppo. Ciò ha portato alla definizione del progetto come tipo di dato astratto ed alla individuazione degli elementi e delle funzionalità di base a supporto della descrizione, organizzazione e controllo del progetto stesso.

Tra gli elementi che caratterizzano il progetto sono stati individuati:

- le *componenti logiche*, che derivano da una suddivisione del progetto in moduli logicamente distinti, quali ad esempio l'interprete ed il compilatore nel progetto di implementazione di un linguaggio di programmazione; ad ogni componente logica corrisponde una parte di file system, e ad ogni stadio del suo sviluppo è associato un numero di versione;
- gli *utenti* che partecipano allo sviluppo del progetto;
- i *sottoprogetti*, che rappresentano una associazione tra componenti logiche ed utenti, nel senso che un sottoprogetto ha l'obiettivo di sviluppare una o più componenti logiche e ad esso contribuiscono uno o più utenti; un sottoprogetto definisce una vista[3] del progetto attraverso la quale vengono specificati i diritti di accesso degli utenti che ad esso sono associati;
- un insieme di *vincoli*, che limitano l'applicabilità delle operazioni sulla descrizione del progetto.

L'evoluzione del progetto avviene attraverso una fase di definizione di questi elementi, ed una serie di fasi di sviluppo di sottoprogetti, ciascuna scandita da un'operazione di inizio (split), e da una operazione di chiusura (merge), che comporta il rilascio di una nuova versione delle componenti logiche sviluppate dal sottoprogetto.

Lo *stato* del progetto è pertanto costituito da una istanza della descrizione del progetto ottenuta attraverso un'operazione di creazione e l'applicazione di una sequenza di operazioni su di essa. Lo stato contiene la descrizione degli utenti, delle componenti logiche, dei sottoprogetti e dei vincoli sulla base dei quali il progetto può evolvere attraverso l'uso delle operazioni, e la descrizione dello stato di avanzamento del progetto determinato dalle operazioni di rilascio di versioni. Esso, quindi, può essere definito come una descrizione ai morsetti del progetto, definita dall' utente e parzialmente aggiornata dal sistema, attraverso la quale quest'ultimo può garantirne un'evoluzione controllata.

La *storia* del progetto è definita dalla sequenza nel tempo degli stati, via via modificati dalle varie operazioni, cioè dalla sequenza di coppie [evento,stato] che si sono susseguite durante il suo sviluppo. Per mezzo della storia è possibile dunque ricostruire tutti gli stati nei quali si è venuto a trovare il progetto durante la sua evoluzione. Questa definizione di storia fa riferimento non soltanto all'evoluzione dei moduli software (vedi ad esempio [1]), che in SCAV sono rappresentati dalle componenti logiche, ma più in generale alla organizzazione del progetto, definita attraverso sottoprogetti, vincoli ed utenti. Ciò consente, come si vedrà in seguito, di utilizzare l'informazione contenuta nella storia anche per scopi diversi dal controllo delle versioni.

[3] Il termine va inteso con il significato usuale attribuitogli nel campo delle basi di dati.

Per poter definire una politica di mantenimento della storia del sistema si è dovuta individuare una nozione di rilevanza delle informazioni contenute nella storia, in riferimento allo stato corrente. Abbiamo pertanto definito quali dati, in SCAV, possono diventare obsoleti e le condizioni che possono causare tale trasformazione. Le informazioni che possono diventare inutili sono relative a:

- la sequenza degli eventi, cioè le istruzioni di SCAV che sono state eseguite durante lo sviluppo del progetto;
- le versioni delle componenti logiche e le versioni dell'intero progetto rilasciate in fase di sviluppo.

Il criterio che è stato seguito è di considerare obsoleto un dato solo quando la sua eliminazione non modifica lo stato corrente del sistema, ed, allo stesso tempo, mantiene *consistente* la storia del progetto; tale caratteristica consente la ricostruzione dello stato del sistema, facendo rieseguire le istruzioni rimaste dopo le eventuali cancellazioni.

L'analisi della storia non si limita al problema delle cancellazioni ma consente il recupero di informazioni concernenti le varie fasi dell'evoluzione del progetto. Il sistema SCAV fornisce una interfaccia, utilizzabile tramite il comando query [3], che consente l'interrogazione della storia, e riporta l'informazione direttamente memorizzata negli stati senza, però, fare elaborazioni o analisi dei dati. Se, per esempio, si volesse conoscere quali sono stati gli utenti che hanno contribuito allo sviluppo di una determinata versione di una componente logica, si dovrebbe reperire tale informazione in maniera analitica: si dovrebbe cioè controllare la sequenza delle istruzioni, vedere quando è cominciato lo sviluppo della versione che interessa e quando è terminato, per poi controllare in questo periodo quali sono stati gli utenti che hanno fatto parte del sottoprogetto a cui è stato affidato lo sviluppo della componente.

L'analisi della storia del sistema è stata quindi riconsiderata con l'obiettivo di fornire al responsabile del progetto degli strumenti per il reperimento di informazioni non elementari, in modo da consentire una maggiore conoscenza sullo sviluppo del sistema, ed, eventualmente, offrire una guida nella scelta delle successive istruzioni da eseguire.

3 LA DESCRIZIONE PER MEZZO DEL CALCOLO DEGLI EVENTI

Come osservato in precedenza, la storia in SCAV può essere vista come una sequenza di coppie [evento, stato] che rappresentano l'evoluzione di un progetto software. Per l'analisi della storia abbiamo preso in considerazione solo la sequenza degli eventi, tralasciando le informazioni relative allo stato, che si possono da essa ricostruire. Questo aspetto sarà discusso più in dettaglio nel seguito; per il momento è sufficiente osservare che la sequenza degli eventi è stata considerata come base delle conoscenze e si è cercato

di individuarne una rappresentazione adeguata. Il formalismo da utilizzare deve consentire la rappresentazione ed il ragionamento circa il tempo ed il Calcolo degli Eventi [12] si è rivelato adatto allo scopo. Come il nome suggerisce, in questo calcolo la nozione di evento è centrale: le descrizioni di eventi implicano l' esistenza di periodi di tempo nei quali certe relazioni sono verificate. Questo Calcolo è stato sviluppato per trattare gli aggiornamenti alle basi di dati, che vengono descritti da eventi; l'effetto della cancellazione esplicita di una informazione è ottenuto definendo la fine di un periodo di tempo nel quale tale informazione è vera.

Nel calcolo degli eventi le regole sono in forma di clausole di Horn e, quindi, direttamente esprimibili in PROLOG; nel seguito esporremo brevemente come tale calcolo è stato sfruttato per la descrizione della storia di SCAV, rimandando alla bibliografia per una sua descrizione completa.

SCAV registra nella storia i comandi nel seguente formato:

operazione / nome-oper / autore / data / num-param / elenco-parametri

Dato che ad ogni comando corrisponde una transizione di stato, ossia un evento, un comando viene tradotto nella seguenti clausole:

```
act(operazione,nome-oper).
actor(operazione,autore).
date(operazione,data).
num(operazione,num-parametri).
object(operazione,lista-parametri).
```

in cui operazione, che in SCAV rappresenta il numero progressivo dell'operazione eseguita, diventa l'identificatore di un evento, e gli altri argomenti vanno a specificare un'insieme di relazioni caratteristiche dell'evento, quali il tipo, l'autore, la data in cui è avvenuto ed i suoi parametri.

Nel calcolo degli eventi si possono specficare periodi di tempo per mezzo dei termini after(E,U) e before(E,U) dove U è la relazione che inizia a valere dopo (prima) l'evento E. Ad essi si fa riferimento per definire l'intervallo di tempo in cui una relazione vale;

```
1 holds(before(E,U)):-terminates(E,U).
  holds(after(E,U)):-initiates(E,U).
```

cioè la relazione U è vera prima (dopo) dell'evento E se questo termina (inizia) la relazione U; le regole initiates e terminates dipendono dal caso particolare. Es.

```
initiates(E,in-pro(X,Y)):-act(E,adduser),
                          object(E,[X]),
                          Y=nil.
initiates(E,in-pro(X,Y)):-act(E,adduser),
                          object(E,[X,Y]).

holds(after(E,in-pro(X,Y)):-initiates(E,in-pro(X,Y)).
```

indica che l'utente X partecipa al progetto dopo l'evento E, se ad esso corrisponde un'istruzione di adduser; appartiene, sempre dopo l'evento E, al sottoprogetto Y se nella lista dei parametri ci sono due elementi, cioè l'istruzione di adduser specifica anche il sottoprogetto.

Per controllare se una relazione è vera durante tutto un intervallo di tempo si definisce la relazione broken.

```
2 broken(E,U,E1):-exclusive(U,U1),
                  holds(after(E2,U1)),
                  E<E2,
                  E2<E1.
  broken(E,U,E1):-exclusive(U,U1), -
                  holds(before(E2,U1)),
                  E<E2,
                  E2<E1.
```

La relazione U non è vera per tutta la durata dell'intervallo che va dall'evento E all'evento E1 se c'è una relazione U1, che non può essere vera contemporaneamente ad U (exclusive), ed è iniziata o terminata da un'evento E2 tale che E<E2<E1. Si noti che i due eventi E ed E1 devono essere istanziati, mentre la relazione exclusive esprime le incompatibilità tra le relazioni, e dipende dalla relazione considerata. Ad esempio

```
exclusive(visitor-programmer(S),project-boss(S)).
```

indica che che i ruoli di responsabile del progetto e visitatore sono incompatibili, ed il goal

```
?-broken(13,project-boss(user),23)
```

è soddisfatto se l'utente user non è responsabile del progetto ininterrottamente dall'evento 13 al 23, cioè se esiste un evento E2, compreso tra 13 e 23, dopo il quale (o prima del quale) user è un semplice visitatore.

Per sapere se una relazione è vera ad un certo istante di tempo si definisce la relazione holdsat.

```
3 holdsat(E,U):-holds(after(E1,U)),
                E1<E,
                not broken(E1,U,E).
  holdsat(E,U):-holds(before(E1,U)),
                E1>E,
                not broken(E1,U,E).
```

la relazione U è vera all'istante prima dell'evento E se esiste un evento E1 dopo (prima) il quale U è vera con E1<E e tra E ed E1 non esiste un evento che la interrompe. Si noti che anche in questo caso E deve essere istanziato. Ad esempio il goal

```
?-holdsat(13,in-pro(user,_))
```

è soddisfatto se l'utente user lavora al progetto al momento dell'evento 13, quindi è vera se esiste un evento E1 che ha inserito user nel progetto prima di E e non esiste nessun evento tra E1 ed E che lo elimina.

Per conoscere l'intervallo di tempo entro il quale una relazione è vera si definisce la relazione during.

```
4 during(E,U,E1):-holds(after(E,U)),
                  holds(before(E1,U)),
                  E<E1,
                  not(broken(E,U,E1)).
```

Si noti che per via della definizione di holds, la relazione during identifica esattamente gli estremi dell'intervallo in cui la relazione è vera e non vale nei suoi sottointervalli. Ad esempio il goal

```
?-during(13,in-subp(user,subp1),23)
```

è soddisfatto se l'utente user è stato associato al sottoprogetto subp1 nel periodo che va dall'evento 13 al 23.

La base delle conoscenze è stata definita utilizzando le relazioni qui illustrate. A partire da esse si può ricostruire lo stato del sistema, nel senso che si può verificare se le relazioni che lo caratterizzano sono soddisfatte.

In SCAV la storia registra sia le operazioni che gli stati per ottenere una maggiore efficienza nel reperimento delle informazioni sullo stato: è infatti troppo complesso da un punto di vista realizzativo e/o dispendioso in termini di tempo di esecuzione, ricavare le stesse informazioni tramite procedure a partire dagli eventi. è stato quindi preferito un maggiore dispendio di memoria a vantaggio di una maggiore velocità di risposta. Un ulteriore miglioramento è stato ottenuto con l'uso di SCCS [15].

L'uso di un formalismo dichiarativo quale il calcolo degli eventi consente di risparmiare la memorizzazione esplicita degli stati senza nessuna difficoltá nella realizzazione, fornendo allo stesso tempo delle prestazioni accettabili. Il motivo di ciò va cercato nel fatto che tramite il calcolo degli eventi è possibile far riferimento alle singole relazioni sullo stato senza doverlo completamente ricostruire e senza nessuna complicazione dell'implementazione.

4 LE FUNZIONALITÀ REALIZZATE

Una volta individuato il formalismo tramite il quale descrivere la base di conoscenze abbiamo studiato la sequenza degli eventi corrispondenti alle istruzioni di SCAV che sono state eseguite durante lo sviluppo di un progetto. L'analisi degli eventi è stata condotta in modo da raggiungere i seguenti obiettivi:

1. Riduzione della descrizione dello sviluppo del progetto al minimo rilevante per lo stato corrente.
2. Analisi delle versioni del progetto al fine di ridurre l'occupazione di memoria nel file system.
3. Introduzione di nuove primitive per l'interrogazione intelligente dell'informazione contenuta nella storia con lo scopo di aumentare gli strumenti di controllo ed organizzazione del progetto.

Questi obiettivi saranno discussi separatamente nel seguito.

4.1 La minimizzazione della descrizione del progetto

Lo scopo principale della minimizzazione della descrizione del progetto è quello di limitare le informazioni contenute nella storia alle sole rilevanti, in modo da focalizzare più facilmente l'attenzione dell'utente e del sistema sui dati importanti e di garantirne un più semplice e veloce reperimento.

Allo scopo di identificare la rilevanza delle informazioni contenute nella storia sono stati esaminati i diversi elementi della descrizione del progetto definiti da SCAV, e cioè i vincoli, gli utenti ed i sottoprogetti.

SCAV fornisce la possibilità di definire una serie di vincoli (Constraints) che descrivono in che modo il progetto può crescere attraverso l'uso delle operazioni, e quindi determinano le modalità di evoluzione del progetto. In particolare i vincoli descrivono per ogni operazione di SCAV quale utente può usarla e in che modo. Esistono funzioni i cui diritti d'uso sono dati non al singolo utente ma all'intero sottoprogetto (cioè a tutti gli utenti del sottoprogetto). Ogni volta che viene tolta ad un utente o ad un sottoprogetto la possibilità di eseguire un'operazione ci si può domandare se essa sia stata sfruttata o meno nel periodo in cui ciò era possibile, ed in caso di risposta negativa si può pensare di eliminare gli eventi riguardanti la sua concessione e rimozione.

Gli utenti di un progetto descritto tramite il formalismo offerto da SCAV hanno un ruolo che dipende dalle possibilità di operare determinate dai vincoli: possono essere semplicemente dei visitatori (non hanno diritto di scrittura su nessuna componente logica), oppure dei progettisti, o avere maggiori possibilità di azione fino a poter modificare i vincoli delle operazioni ed influenzare in tal modo la politica di evoluzione del progetto. Per diversi motivi si può decidere di rimuovere un utente da un sottoprogetto: ad esempio perché la componente logica da esso sviluppata ha raggiunto uno stadio di avanzamento tale da richiedere una quantità minore di risorse. In questo caso va analizzato il contributo allo stato corrente del sistema, con riferimento al sottoprogetto, da parte dell'utente che si intende rimuovere e nel caso risulti non significativo si possono dimenticare le informazioni inerenti la sua appartenenza al sottoprogetto. Considerazioni analoghe valgono nel caso in cui un utente viene rimosso dall'intero progetto.

Il sottoprogetto realizza la connessione tra gli utenti e le componenti logiche: è una particolare vista del progetto, tramite la quale viene normalmente assegnato l'accesso in lettura a tutto il progetto, e in scrittura ad una o più componenti logiche. Anche i sottoprogetti possono essere rimossi, ad esempio in caso di riconfigurazione della struttura del progetto. Eliminare un sottoprogetto ha come ovvia conseguenza quella di rimuovere tutte le sue associazioni con gli utenti e tutti i diritti ad esso direttamente assegnati, e quindi si applica quanto detto nei punti precedenti. Se inoltre, si verifica che la presenza del sottoprogetto è diventata ininfluente perché gli utenti ad esso associato non hanno sviluppato nessuna versione ancora rilevante del progetto o di qualche componente logica, l'esistenza del sottoprogetto può essere completamente dimenticata.

4.2 La minimizzazione delle versioni

In SCAV l'evoluzione del progetto è determinata dal rilascio di nuove versioni delle singole componenenti logiche o dell'intero progetto. La descrizione dei meccanismi di rilascio di versioni porta ad una strutturazione del numero di versione in quattro campi: il primo di essi è legato all' operazione di Release, cioè si incrementa di uno ogni volta che viene rilasciata una versione stabile del progetto; il secondo è invece legato alle ope-

razioni di rilascio di versioni di componenti logiche; il terzo e il quarto sono infine legati ai conflitti che si possono generare in fase di sviluppo quando l'inizio dell'attività di un sottoprogetto fa riferimento ad una versione del progetto diversa dall'ultima rilasciata (una descrizione più dettagliata si può trovare in [7]).

In effetti una versione del progetto è l'unione delle componenti logiche in cui esso è strutturato considerate ognuna ad una certa versione: è, dunque, un'associazione tra un nome ed una lista di componenti logiche con il relativo identificatore di versione costituito da una quadrupla di numeri.

Il rilascio di una versione dell'intero progetto definisce un insieme di componenti logiche cui fare riferimento in modo globale nelle fasi successive di sviluppo del progetto.

Possiamo, quindi, dare le seguenti definizioni di versione obsoleta:

Una versione del progetto può essere considerata obsoleta se è stata rimossa, ossia non è più utilizzabile per la creazione di una nuova vista del progetto, e se non è servita nella creazione di una vista per un'altra versione ancora utile. Quando una versione del progetto viene considerata obsoleta si possono dimenticare le relative informazioni: chi l'ha rilasciata e quando, quali erano le versioni delle sue componenti logiche.

Poiché le componenti logiche vengono sviluppate separatamente una versione di una componente logica si considera obsoleta se non appartiene né ad alcuna versione del progetto né è servita come punto di partenza di altre versioni.

Illustriamo questo concetto con un esempio. Supponiamo che allo stato corrente del sistema appartenga la versione del progetto VERS1 composta dalle seguenti versioni di componenti logiche:

COMP1 0.1.0.0 COMP2 0.1.0.0

Supponiamo, inoltre, che ci siano due sottoprogetti, che chiamiamo SUB1 e SUB2 e che siano associati, rispettivamente, a COMP1 e COMP2. Se SUB1 viene attivato prendendo come vista VERS1 e successivamente rilascia (con un'operazione chiamata Merge) la versione del progetto Vers2, questa sarà composta da

COMP1 0.2.0.0 COMP2 0.1.0.0

Analogamente può essere rilasciata la versione Vers3 costituita da

COMP1 0.3.0.0 COMP2 0.1.0.0

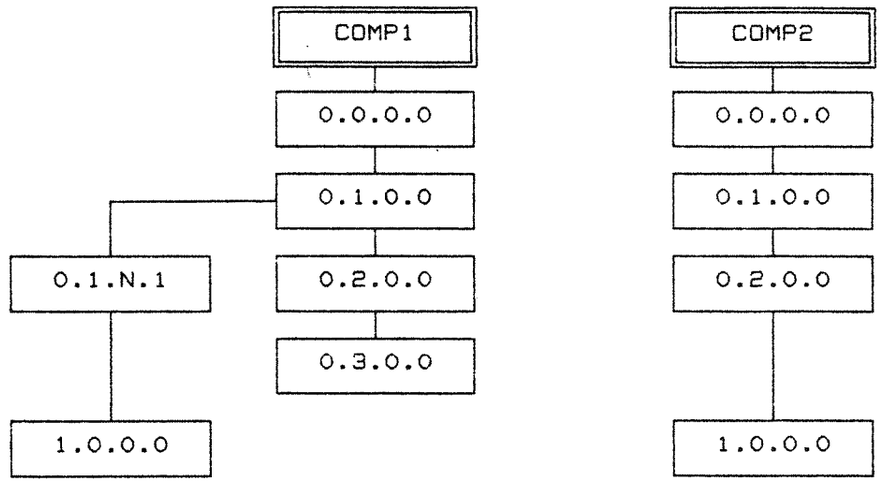
Se a questo punto SUB1 viene attivato facendo riferimento a Vers1, la versione rilasciata in seguito da questo sottoprogetto (chiamiamola Vers4) è costituita da

COMP1 0.1.N.1 COMP2 0.1.0.0

N è l'identificatore del sottoprogetto; in questo caso viene generata una diramazione dell'albero delle versioni, espressa tramite il terzo e quarto elemento del numero di versione, e causata dal riferimento ad una versione diversa dall'ultima rilasciata. Supponiamo che nel frattempo SUB2 sia stato attivato su Vers1 e abbia rilasciato la versione del progetto Vers5 definita da

COMP1 0.1.0.0 COMP2 0.2.0.0

Se, a questo punto, si vuole creare una versione del progetto che costituisce un punto di sincronizzazione per tutti i componenti del progetto si deve dare un comando di release. Tale versione può essere costruita tanto sulla COMP1 0.1.N.1, che sulla COMP1 0.3.0.0. Consideriamo qui il primo caso: la situazione può essere illustrata graficamente nel seguente modo.



Rimuoviamo Vers1 e Vers3 e vediamo quali versioni di COMP1 sono diventate obsolete.

0.3.0.0 è obsoleta perché non appartiene a nessuna versione presente nello stato attuale e non è stata usata nella creazione di nessuna vista

0.2.0.0 non è obsoleta perché, anche se non è stata usata nella creazione di una vista che ha portato allo sviluppo di versioni di COMP1 ancora utili (0.3.0.0 è stata rimossa), appartiene a Vers2

0.1.0.0 non è obsoleta perché è servita per lo sviluppo di 0.1.N.1 e 0.2.0.0

Gli altri casi sono evidenti.

In definitiva possiamo dire che devono essere conservate tutte le versioni appartenenti al cammino dalla radice alle foglie che rappresentano versioni appartenenti allo stato corrente del sistema. Considerare obsoleta una versione di una componente logica significa poter dimenticare le informazioni ad essa relative, ma, soprattutto, la sua rappresentazione nel file system; quest'ultima circostanza ha, a differenza degli altri casi, un notevole impatto sull'occupazione di memoria.

Gli eventi che possono rendere obsolete versioni del progetto e di componenti logiche non sono solo di tipo rimozione, come ad esempio la rimozione di una versione del progetto, ma anche istruzioni di rilascio di versioni (release, merge) che possono, come effetto collaterale, rendere inutili versioni di componenti logiche: questo può succedere perché il sistema gestisce una serie di versioni predefinite del progetto che vengono aggiornate dalle operazioni di rilascio di versioni.

4.3 La propagazione delle cancellazioni

L'analisi della storia fatta ad un certo istante della evoluzione del progetto non è definitiva. Infatti è possibile che l'individuazione di eventi obsoleti porti a riconsiderare con risultati diversi parti della storia analizzate in precedenza.

Consideriamo, ad esempio, la seguente sequenza di istruzioni:

1 Aggiunta dell'utente User

2 Aggiunta (da parte di User) della versione Vers

3 Rimozione di User

Gli eventi 1 e 2 non appartengono ai tipi di istruzioni che possono portare a delle cancellazioni; 3 vi appartiene ma User non può essere dimenticato perché ha aggiunto una versione ancora utile. Se in seguito viene eseguito il comando

4 Rimozione di Vers

e Vers diventa obsoleta, gli eventi 2 e 4 possono essere dimenticati; a questo punto anche 1 e 3 possono essere dimenticati.

Lo studio della propagazione delle cancellazioni ha portato ad osservare che la rimozione di un evento non può comportare la mancata rimozione di un altro evento. In altre parole la cancellazione di un evento può eventualmente creare le condizioni per eseguire nuove cancellazioni, ma non può alterare le condizioni che già rendono possibili delle cancellazioni.

Questa proprietà, che vale nel caso delle operazioni definite dal sistema SCAV, ha come conseguenza che l'insieme delle cancellazioni eseguite non dipende dall'ordine con cui queste vengono eseguite.

Si è quindi definito un algoritmo di propagazione delle cancellazioni in grado di realizzare efficientemente tale processo; esso viene qui descritto informalmente.

È stato individuato l'insieme di eventi che possono portare a delle cancellazioni, tipicamente operazioni di rimozione di entitá del progetto o di rilascio di versioni.

L'algoritmo esegue una scansione degli eventi seguendo l'ordine temporale, alla ricerca di un evento che puó portare ad una cancellazione; sia E l'evento trovato.

L'individuazione dell'evento cancellabile avviene analizzando all'indietro gli eventi a partire da E; sia E1 l'evento da cancellare tale che $E1 < E$.

Le eventuali cancellazioni causate dalla rimozione dell'evento E1 vengono cercate scandendo all'indietro gli eventi nell'intervallo tra E1 ed E. La ricerca è limitata ad E1 in quanto per gli eventi precedenti E1 non sono mutate le condizioni rispetto all'analisi fatta in precedenza. In caso di successo il processo di propagazione viene invocato ricorsivamente. In caso di fallimento l'algoritmo procede con l'analisi dell'evento immediatamente seguente E.

La procedura può essere ulteriormente snellita se si tiene conto del fatto che cancellazioni di eventi diversi portano ad esaminare diverse possibilità di propagazione, ad esempio la rimozione di un'assegnazione di diritti d'uso, non può avere ripercussioni dirette sulla cancellazione di versioni obsolete. In questo modo vengono ridotte al minimo le parti della storia che devono essere analizzate per completare il processo di cancellazione.

Si noti tuttavia che da un punto di vista pratico può essere comodo adottare una strategia batch per la rimozione delle cancellazioni, ossia attivare la procedura di compattazione ad intervalli, ad esempio settimanali. In questo caso, se si accetta che l'attivazione della procedura di cancellazione possa lasciare degli eventi obsoleti, che saranno eventualmente rimossi dalle successive attivazioni, il problema della propagazione delle cancellazioni può essere del tutto ignorato.

4.4 L'interrogazione della storia

La creazione di una rappresentazione esplicita della descrizione del progetto, sviluppata inizialmente allo scopo di organizzare convenientemente la storia del progetto, ha successivamente portato a considerare la possibilitá di migliorare il sistema di consultazione della storia con l'obiettivo di fornire informazioni utili, ed eventualmente una guida, al controllo ed organizzazione del progetto.

Al momento, questa parte del lavoro è in fase di analisi, con alcuni esempi di realizzazione di carattere per lo più esplorativo. Nel seguito cerchiamo di delineare attraverso alcuni esempi i problemi che possono essere affrontati. Sottolineiamo ancora, come l'uso di una metodologia di tipo dichiarativo consenta di giungere ad una realizzazione in tempi estremamente rapidi.

Esempi di informazioni reperibili dalla storia attraverso semplici elaborazioni sono l'insieme dei progettisti che hanno contribuito allo sviluppo di una componente logica, o di una versione dell'intero progetto. Un'analisi di questo tipo può essere approfondita facendo riferimento al ruolo avuto dai progettisti durante lo sviluppo della versione di interesse.

Sempre facendo riferimento alle versioni del progetto e delle componenti logiche può essere utile sapere come è possibile renderne obsoleta qualcuna, oppure perché una certa versione è ancora non obsoleta, in modo da pianificare le operazioni di cancellazione esplicita. Si ricorda a questo proposito che l'uso del meccanismo di rilascio di versioni dell'intero progetto serve anche a tener traccia di operazioni di distribuzione, per cui una componente logica può essere ancora utile ad esempio perché parte di una versione rilasciata ad un acquirente.

Un altro aspetto di interesse riguarda l'effetto delle definizioni dei vincoli, per i quali è possibile individuare incompatibilità, sovrapposizioni, mancanze, etc. L'uso della funzione di definizione dei vincoli non è semplice e, può risultare molto utile sapere le conseguenze di una assegnazione. Un esempio significativo in questo senso può essere quello di volere conoscere quali utenti siano in grado di abilitare un determinato utente X ad eseguire una certa operazione Y; le risposte possono essere diverse:

a l'utente Usr può direttamente dare la possibilità ad X di fare l'operazione Y;

b l'utente Usr1 può dare la possibilità a User2 di modificare i vincoli dell'operazione Y dopodiché User2 potrebbe dare ad X il diritto di fare Y;

c l'utente User1 può dare la possibilità a User3 di modificare i vincoli dell'operazione "modifyconstr" per quanto riguarda Y nei confronti di User2 il quale, quindi, potrebbe essere abilitato (da User3) a modificare i vincoli di Y e quindi dare la possibilità ad X di fare Y.

Molti degli esempi accennati, ed in particolare l'uso del sistema di vincoli, suggeriscono un approfondimento della nozione di ruolo, già presente nel sistema SCAV, per la definizione dei diritti di un utente al momento del suo ingresso nel progetto. In realtà un uso appropriato dei ruoli può essere utile non solo da un punto di vista definitorio ma anche per quanto riguarda il riconoscimento del ruolo che un utente ha avuto nello sviluppo di un progetto sulla base dei vincoli e delle operazioni eseguite.

5 CONCLUSIONI

In questo lavoro è stato considerato il problema di memorizzare ed utilizzare le informazioni relative all'evoluzione di un progetto software facendo riferimento alla descrizione sviluppata tramite il sistema SCAV. È stato utilizzato il Calcolo degli eventi come formalismo per rappresentare la storia del sistema determinata dall'esecuzione delle operazioni eseguite da SCAV. Sulla base di questa rappresentazione sono state definite un insieme di funzionalità per l'individuazione delle informazioni rilevanti della storia, ossia degli eventi che determinano lo stato corrente del sistema, e per l'utilizzo delle informazioni della storia al fine di guidare le attività di pianificazione ed organizzazione del progetto.

L'analisi della storia di un sistema ha importanza in una vasta classe di sistemi, e l'obbiettivo iniziale di questo lavoro è stato quello di studiarne una rappresentazione specifica nel caso dello sviluppo di progetti software, cercando di ottenere una descrizione minimale, con particolare riguardo alle parti la cui eliminazione ha un impatto significativo sull'occupazione di memoria dei files del progetto.

Tra i risultati del lavoro di particolare interesse è la verifica di alcune caratteristiche della modellazione di un progetto software definita dal sistema SCAV adottando una metodologia di tipo dichiarativo. Questa scelta ha portato alla costruzione di una base delle conoscenze relative alla evoluzione di un progetto software, sulla quale definire strumenti di supporto alla organizzazione ed al controllo del suo sviluppo. A questo proposito il Calcolo degli eventi si è rivelato uno strumento adeguato, consentendo la ricostruzione dello stato del sistema ad un certo istante della sua evoluzione, in modo estremamente flessibile. Come conseguenza di ciò è stato possibile sviluppare in tempi molto rapidi alcune funzioni che manipolano le informazioni contenute nella storia.

L'opportunità di utilizzare un formalismo logico per la descrizione di un progetto e della sua evoluzione viene discussa da Asirelli et al. [1], con motivazioni analoghe. Nel nostro caso però, l'uso di un formalismo di tipo logico si riferisce alla costruzione e manipolazione di una base delle conoscenze cui l'utente ha accesso attraverso le funzionalità del sistema SCAV, affiancate da quelle qui descritte relative alle informazioni contenute nella storia, mentre nel lavoro citato l'utente fornisce la descrizione del sistema direttamente in un formalismo logico. Il sistema qui descritto offre pertanto un ulteriore livello di astrazione relativo alla modellizzazione dell'evoluzione del progetto, cui corrisponde la realizzazione di un insieme di comandi che consentono all'utente di operare più agevolmente sulla descrizione del progetto.

Allo stato attuale del lavoro, le funzionalità che operano la compattazione della storia sono state completamente definite e realizzate, mentre per quanto riguarda l'utilizzo della storia per la realizzazione di strumenti di supporto alla organizzazione del progetto, sono state per ora sperimentate solo alcune delle possibilità offerte dalla rappresentazione. In particolare si intende sviluppare la nozione di ruolo facendo riferimento a degli schemi

di definizione dei vincoli predefiniti (successivamente anche definibili dall'utente) in modo che il sistema sia in grado di riconoscere automaticamente il ruolo di un utente che partecipa ad un sottoprogetto. La disponibilità di informazioni di questo tipo consentirà di approfondire lo studio di strumenti di pianificazione all'attivita' di organizzazione e gestione di progetti software.

Bibliografia

[1] Asirelli P., Giannini F., Grifoni E., Inverardi P., La programmazione logica in ambienti di sviluppo software: un data base di progetto logico, Atti del Convegno GULP87, pp. 173-182, Torino, Maggio 1987.

[2] Cecchini, M., Lucchesi L., Implementazione di SCAV Sistema di controllo Accessi e Versioni, Rapporto Tecnico PRISMA 4/85, Sipe Optimation, Roma, Ottobre 1985.

[3] Cecchini, M., Lucchesi L., SCAV Reference Manual, Rapporto Tecnico PRISMA 1/86, Sipe Optimation, Roma, Gennaio 1986.

[4] Cheatham, T., An Overview of the Harvard Program Development System, in Software Engineering Environments, ed. H. Hunke, North-Holland, 1981.

[5] Claybrook, B. G., Claybrook A. M., Williams J., Defining Database Views as Data Abstractions, IEEE Transactions on Software Engeneering, vol. SE-11(1), pp. 3-14, 1985.

[6] Dariol, D., Nardi D., Il Sistema di Controllo Accessi e Versioni, Rapporto Tecnico PRISMA 1/84, Sipe Optimation, Roma, Ottobre 1984.

[7] Dariol, D., Nardi D., Una proposta di realizzazione per il Sistema di Controllo Accessi e Versioni, Rapporto Tecnico PRISMA 1/85, Sipe Optimation, Roma, Gennaio 1985.

[8] Dariol, D., Nardi D., Un Sistema per il Controllo di Accessi e Versioni, in Atti del Congresso AICA '85, pp. 197-321, Roma, Ottobre 1985.

[9] Huber-Bachrich, A., L'uso della storia in SCAV, Tesi di Laurea, Università di Roma "La Sapienza", Luglio 1987.

[10] Fraser C., Myers E., An editor for revision control, ACM TOPLAS, 9:2, pp. 277-295, 1987.

[11] Katz, R. H., Lehman T. J., Database Support for Versions and Altenatives of Large Design Files, IEEE Transactions on Software Engeneering, vol. SE-10(2), pp. 191-200, 1984.

[12] Kowalski, R., Sergot M., A Logic-based Calculus of Events, in New Generation Computing, vol. 4:1, pp. 67-95, 1986.

[13] Lampson, B. W., Schmidt E. E., Organizing Software in a Distributed Environment, in Proceedings on Programming Languages issues in software systems, ACM SIGPLAN, 1983.

[14] Notkin, D., The GANDALF Project, The Journal of Systems and Software, vol. 5, pp.91-105, 1985.

[15] Rochkind, M. J., The Source Code Control System, IEEE Transactions on Software Engeneering, vol. SE-1(4), pp. 364-370, 1975.

[16] Tichy, W. F., A System for Version Control, Software Practice and Experience, vol. 15(7), pp. 637-654, 1985.