POLITECNICO DI MILANO
Facoltà di Ingegneria
Dipartimento di Elettronica

FORMALIZZAZIONE DI UN PROCESSO SOFTWARE INDUSTRIALE: RISULTATI E VALUTAZIONI

Relatore:
Prof. Carlo GHEZZI

Correlatori:
Ing. Maurizio LOI
Dott. Sergio BANDINELLI

Tesi di Laurea di:
Gian Pietro PICCO
matr. n° 587293

Anno Accademico 1992/93

Ai miei genitori

Ringraziamenti

Desidero ringraziare prima di tutto il Prof. Carlo Ghezzi, non soltanto per avermi dato la possibilità di svolgere questo lavoro di tesi ma anche e soprattutto per essersi sempre mostrato estremamente disponibile nei miei confronti, prima in veste di docente e poi in veste di relatore.

Un ringraziamento doveroso ma sentito va all'Ing. Maurizio Loi il quale, animato da reale interesse verso il mio lavoro, mi ha seguito fin dall'inizio, prima fornendomi tutto l'aiuto necessario al fine di acquisire le informazioni richieste dal progetto e poi procedendo ad attente revisioni del modello, riuscendo a conciliare una grande serietà professionale con altrettanta cordialità.

Un ringraziamento anche al Dott. Sergio Bandinelli, che ha fornito preziosi consigli soprattutto nelle fasi di impostazione iniziale del lavoro e con il quale è stato tutto sommato anche divertente lavorare.

Poiché questo lavoro di tesi è stato svolto all'interno della quinta edizione del Master in Tecnologie dell'Informazione del CEFRIEL, è mio dovere ringraziare i responsabili di tale organizzazione per avermi offerto tale possibilità: oltre a ciò, vorrei in particolare ringraziare il Prof. Fuggetta per le preziose indicazioni bibliografiche e relativo materiale fornitomi durante il lavoro.

La realizzazione di una tesi, alla fine di un ciclo di studi, non è solo un fatto puramente tecnico; desidero pertanto ringraziare soprattutto i miei genitori, i quali (e non è retorica) hanno fatto grandi sacrifici per permettermi di raggiungere questo risultato. A loro un ringraziamento del tutto particolare.

Un pensiero va agli amici del CEFRIEL, che hanno trascorso con me, giorno per giorno, praticamente tutto il periodo della tesi: un saluto va quindi agli amici dell'area Software Engineering, Fabiano, Guido, Marco M., Marco S., Paola, Paolo e Stefano, e dell'area Factory Automation, Marcella, Anna, Gianluigi e Margherita.

E infine, ultima nell'ordine ma non nei pensieri, un ringraziamento particolare va a Cristina, amica da sempre ed oggi a me particolarmente vicina, che ha saputo scuotermi, fiduciosa, nei momenti di sconforto e partecipare, entusiasta, ai momenti di gioia.

A tutti, grazie.

Indice

Capitolo 1

Introduzione

Le applicazioni software sono ormai diffusamente presenti in ogni settore della vita quotidiana, ricoprendo anche ruoli assai critici: poiché un'applicazione software è di fatto un prodotto ingegneristico, facendo un'analogia con altri settori dell'ingegneria verrebbe istintivo pensare che le applicazioni software in generale, e le applicazioni critiche in particolare, siano adeguatamente supportate da un appropriato e consolidato processo di produzione, comprensivo non solo della descrizione e della pianificazione delle attività che devono essere portate a termine, ma anche delle metodologie, degli strumenti e dei criteri da utilizzare sia per lo sviluppo che per la verifica della validità del prodotto.

Nella pratica industriale invece, ciò non si verifica: un prodotto software soddisfacente in termini di robustezza, correttezza ed efficienza, è ancora in gran parte frutto delle capacità tecniche degli sviluppatori e delle capacità organizzative dei manager di progetto. In altre parole, l'attività di produzione del software può ancora essere considerata in gran parte un'attività di tipo *artigianale*: al contrario, i requisiti imposti dalla diffusione e dalla criticità delle applicazioni generano la necessità di una migrazione verso una produzione del software di tipo *industriale*.

Gli approcci alla gestione del processo di produzione del software ispirati dalle applicazioni ingegneristiche tradizionali sono destinati a fallire, in quanto il software ha caratteristiche peculiari che lo distinguono in maniera netta dagli altri prodotti della tecnologia vincolandone di conseguenza il processo di sviluppo. Ad esempio, la produzione del software è un processo soltanto parzialmente automatizzabile, dal momento che è perlopiù frutto dell'esercizio delle capacità intellettuali dell'individuo; inoltre, è un processo che deve essere modificato frequentemente, contrariamente a quanto accade in settori più tradizionali, per far fronte a cambiamenti dell'ambiente di sviluppo, delle metodologie o addirittura delle fasi del processo, a causa dell'instabilità dei requisiti richiesti per le applicazioni.

Gli strumenti formali, i criteri, le metodologie proposti dall'Ingegneria del Software fino ad oggi hanno contribuito a rendere più efficace e libero da errori il lavoro degli sviluppatori, ma fornendo un tipo di contributo "locale" al problema della gestione del processo di sviluppo del software, migliorando l'operatività singola degli attori del processo ma lasciando sostanzialmente inalterata la loro capacità di cooperare con gli altri attori, in mancanza di una struttura (la descrizione del processo) che possa guidarne le azioni. Ad esempio, il problema generato dalla gestione di un progetto voluminoso in termini di linee di codice è alleviato da tecniche tradizionali dell'Ingegneria del Software, quali la modularizzazione, l'information hiding e, più recentemente, le tecniche di progettazione orientate agli oggetti: ciononostante, tali tecniche non risolvono i problemi di cooperazione, comunicazione, gestione delle risorse umane, i quali sono da molti addirittura considerati *il* problema del processo software.

Le tipologie di problemi generate dal processo di sviluppo del software sono fortemente sentite sia dalle industrie, le quali devono organizzare la produzione di software in modo da fronteggiare tali problemi, che dai committenti delle applicazioni, che desiderano ovviamente un prodotto finale il più possibile affidabile.

Questo ultimo aspetto ha una rilevanza assai elevata soprattutto per quanto riguarda le applicazioni di tipo militare: in questo campo, grazie soprattutto ai grossi finanziamenti in gioco ed ai pesanti interessi politici, sono stati prodotti un certo numero di standard e documenti guida per l'individuazione e la realizzazione delle varie fasi del ciclo produttivo del software, nonché per l'organizzazione di tali fasi all'interno di una cornice globale, costituita dal processo. L'emissione di tali standard ha posto vincoli ben definiti alle industrie che operano in tali settori, in quanto le commesse sono spesso subordinate a certificazioni e verifiche volte ad assicurare che il processo di sviluppo adottato dalle industrie che si propongono per la realizzazione dell'applicazione sia congruente con gli standard, dei quali alcuni sono di portata internazionale.

Il mondo accademico, ma dunque anche quello industriale, hanno preso in seria considerazione il problema, dando luogo ad un nuovo e vivace filone di ricerca la cui nascita si può approssimativamente far risalire alla pubblicazione del lavoro di Osterweil [Ost87], nel 1987.

La ricerca relativa allo studio del processo software può essere divisa in due grandi aree [ABGM92], delle quali una incentrata sugli aspetti manageriali e l'altra sugli aspetti tecnici.

L'area di ricerca che si occupa degli *aspetti manageriali* mira al miglioramento delle attuali pratiche industriali concentrandosi sugli aspetti di tipo organizzativo. L'esempio più rappresentativo in tal senso è costituito dal lavoro svolto al Software Engineering Institute (SEI) della Carnegie Mellon University,che ha portato alla definizione di un modello di maturità delle aziende produttrici di software, classificabili in cinque livelli a seconda della loro capacità di tenere sotto controllo il processo software.

La seconda area si occupa invece degli *aspetti tecnici* coinvolti nella descrizione e nel supporto all'esecuzione del processo software. Uno dei concetti di base di quest'area di ricerca è la presa di coscienza del fatto che non esiste *il* processo software, tale da essere considerato universale e pertanto automatizzato una volta per tutte e replicato per le varie applicazioni, bensì il processo, ed l'ambiente che lo supporta, devono essere adattati allo sviluppo di ogni nuova applicazione.

All'interno di quest'ultima area, spesso identificata come *process modeling*, le notazioni usate per rappresentare il processo software giocano un ruolo essenziale: utilizzando un formalismo per la rappresentazione si è ad esempio in grado di migliorare la comunicazione tra le persone coinvolte nel processo, sia a livello dei tecnici che a livello del management.

Le proposte in letteratura in tal senso sono numerose e partono dagli approcci più svariati, spesso cercando di trasferire e riutilizzare in qualche modo l'esperienza maturata nella descrizione del prodotto software nel processo software; in particolare è possibile identificare *almeno* i seguenti tre tipi di approcci:

- *Linguaggi di programmazione di processo:* il lavoro più rappresentativo in tal senso è quello di Osterweil [Ost87], il quale ha proposto un estensione del linguaggio di programmazione Ada con caratteristiche adatte alla modellazione del processo. Tale lavoro ha costituito un punto di riferimento per tutto il lavoro successivo sul process modeling.

- *Sistemi basati sulla conoscenza:* derivano dalle esperienze vicine alla disciplina dell'Intelligenza Artificiale ed impiegano tecniche tipiche dei sistemi esperti per guidare l'esecuzione del processo a partire dai dati del medesimo, interpretati dalle regole inserite nella base di conoscenza.

- *Formalismi per la specifica di sistemi reattivi:* si basano sull'idea che un processo software sia analogo ad un sistema in tempo reale, dal momento che deve sottostare a precisi vincoli temporali costituiti dalle scadenze di consegna del prodotto finale e dei documenti relativi ad esso. Pertanto, cercano di costruire dei formalismi utilizzabili per la modellazione del processo estendendo i formalismi già in uso per la specifica dei sistemi in tempo reale, quali ad esempio le reti di Petri e gli statecharts.

Questi ed altri tipi di approcci hanno generato un gran numero di proposte in letteratura, nessuna delle quali però si è finora imposta in maniera decisiva rispetto alle altre. Una delle ragioni di ciò va ricercata nel fatto che esistono assai poche esperienze di utilizzo di questi formalismi per la modellazione di processi software *reali*, cioè effettivamente eseguiti *in ambiente industriale*.

Una delle poche esperienze con una certa risonanza all'interno della comunità scientifica è stata quella del SEI, la quale ha utilizzato lo strumento commerciale STATEMATE, nato per la specifica di sistemi in tempo reale mediante un estensione degli automi a stati finiti, per la formalizzazione del processo di modifica della documentazione tecnica relativa al progetto degli F-14 ed F-16 da parte dell'US Air Force [KH89a].

Questo lavoro di tesi si inserisce all'interno dell'area di ricerca interna all'Ingegneria del Software relativa al process modeling, proponendo una esperienza di formalizzazione di una parte del processo di sviluppo del software utilizzato all'interno di una nota industria italiana del settore (Italtel) e quindi inserendosi all'interno della diffusa necessità di casi di studio reali di applicazione dei concetti del process modeling.

Il lavoro si è concentrato sul processo utilizzato all'interno della divisione Italtel che si occupa della realizzazione di applicazioni di tipo militare per il Ministero della Difesa italiano, la cosiddetta Italtel Business Unit Telecommunication Defense (BUTD). Tale divisione realizza in generale sistemi *embedded*, cioè costituiti dall'integrazione di componenti hardware e software: la nostra attività ha preso in esame soltanto gli aspetti relativi al software.

Italtel BUTD Software possiede già una descrizione del processo, contenuta nel cosiddetto Manuale del Sistema di Qualità Software: tale descrizione è però per la stragrande maggioranza redatta in linguaggio naturale, e pertanto soffre delle limitazioni intrinseche a tale rappresentazione, cioè ambiguità ed inconsistenze, impossibilità di automatizzazione dei compiti e di analisi del processo a partire dalla descrizione, e così via.

La nostra formalizzazione si è basata sulla descrizione contenuta nel Manuale di Qualità integrata con le informazioni emerse dai frequenti colloqui con l'esperto della gestione del processo, Ing. Maurizio Loi.

Il modello è stato dunque costruito sulla base delle informazioni così acquisite, impiegando come formalismo per la sua rappresentazione il linguaggio SLANG [1], basato su una estensione ad alto livello delle reti di Petri [GMMP91], il quale costituisce anche il linguaggio di modellazione del processo attorno al quale è costruito l'ambiente di supporto al processo di sviluppo del software SPADE (Software Process Analysis, Design and Enactment) attualmente sviluppato presso il CEFRIEL ed il Politecnico di Milano ([BS93]).

A questo proposito, si deve evidenziare il fatto che questo lavoro di tesi è stato svolto nell'ambito della quinta edizione del Master in Tecnologie Informatiche del CEFRIEL [2], all'interno del quale è stato tra l'altro possibile definire la collaborazione con Italtel.

Gli obiettivi definiti per il lavoro sono i seguenti:

- La stesura di un modello SLANG quanto più possibile completo e consistente di una parte significativa del processo di sviluppo del software in Italtel BUTD.

- La verifica della adeguatezza di SLANG come linguaggio per la descrizione del processo di sviluppo del software: in altre parole, la segnalazione di costrutti e meccanismi inadeguati a modellare specifiche situazioni all'interno del processo, insieme ad eventuali proposte di soluzione.

- La sperimentazione nel campo della modellazione del processo software, in termini di acquisizione [3] delle informazioni relative al processo e

[1] SPADE LANGuage.

[2] Centro per la Ricerca e la Formazione in Ingegneria dell'Informazione

[3] Un termine inglese che rende maggiormente l'idea è *elicitation*, il quale comprende sia la fase di acquisizione della conoscenza, spesso non palesemente descritta in procedure ma nascosta nella pratica quotidiana, sia la fase di sistematizzazione e trasferimento della

costruzione del modello.

Il modello SLANG ha preso in esame una parte cruciale all'interno della gestione del processo, e cioè la *gestione delle anomalie*, ovvero la parte di processo che si occupa di far fronte alla necessità di modifiche a componenti del prodotto software che presentano aspetti non corretti. La criticità di tale gestione risiede nella necessità di operare sugli elementi in maniera ordinata e regolamentata in modo da evitare modifiche incontrollate, le cui differenze rispetto all'originale ed i cui scopi siano in seguito difficilmente ricostruibili.

Per realizzare efficacemente ciò è necessario impostare un sistema di gestione delle configurazioni e delle versioni dei componenti che costituiscono il prodotto software, in altre parole realizzare ciò che in letteratura viene comunemente chiamato Controllo di Configurazione (*Configuration Management*). Poiché quest'ultimo aspetto è intimamente collegato con la gestione delle anomalie e in più riveste notevole importanza anche per il resto della gestione del processo, è stato incluso nella parte di processo da modellare.

Pertanto il nostro modello è costituito da una specifica completa degli aspetti di gestione delle anomalie e della configurazione del prodotto software. In più, è fornita anche una modellazione parziale della gestione delle risorse umane e della effettuazione delle attività di revisione sul prodotto software, anche se limitata soltanto agli aspetti coinvolti nella parte di processo definita precedentemente.

L'attività di modellazione ha effettivamente evidenziato alcune lacune tra i costrutti forniti in SLANG, i quali in alcuni casi (pochi per la verità) si sono mostrati inadeguati alla rappresentazione di determinati aspetti del processo. In alcuni casi è stato anche possibile identificare delle soluzioni adottabili per rendere adeguati tali costrutti allo scopo che si prefiggono.

Infine, si è anche evidenziata una sequenza preferenziale di attività che vanno svolte durante il lavoro di acquisizione e modellazione con SLANG, in modo da poter organizzare secondo una metodologia, per quanto definita in maniera informale e sulla base di un'unica esperienza, eventuali lavori futuri.

La collaborazione diretta con Italtel ha inoltre prodotto effetti collaterali positivi anche sulla descrizione del processo contenuta nel Manuale di Qualità, rilevandone ambiguità, inconsistenze, inadeguatezze e margini di miglioramento, tanto da poter classificare questi come risultati significativi al pari di quelli provenienti dal raggiungimento degli obiettivi sopra elencati.

---

conoscenza così acquisita in un modello.

Pertanto, con riferimento proprio alla lista di obiettivi definiti all'inizio del lavoro, ci pare di poter affermare che tali obiettivi siano stati raggiunti, essendo tale convinzione rafforzata dall'interesse suscitato all'interno di Italtel e di altre aziende partner all'interno del CEFRIEL dai risultati ottenuti dal nostro lavoro come esempio di quelli ottenibili mediante l'impiego del formalismo SLANG.

La struttura della tesi è la seguente: la Parte I approfondisce gli aspetti relativi al problema generale della gestione del processo software, e alla necessità di una sua modellazione formale: in particolare, sono discussi i requisiti che dovrebbero essere soddisfatti da un formalismo per la rappresentazione del processo software. La Parte II contiene una presentazione praticamente completa dei concetti, dei costrutti e dei meccanismi caratteristici del formalismo SLANG, impiegato per la stesura del modello. Nella Parte III viene illustrato il Manuale del Sistema di Qualità Italtel BUTD, su cui si è basata gran parte dell'attività di modellazione: in particolare, il Cap. 4 ne introduce i concetti generali, le finalità e l'organizzazione, i Cap. 5 e 6 descrivono le sezioni del Manuale che hanno come oggetto le parti incluse nel modello, il Cap. 7 presenta le tipologie di problemi incontrati nell'uso del Manuale di Qualità come punto di partenza e base per la modellazione e il Cap. 8 illustra le parti di processo non definite all'interno di tale Manuale ma che verranno incluse prossimamente in quanto frutto dei suggerimenti nati dalla collaborazione con Italtel. Queste ultime parti sono già incluse all'interno del modello, il quale costituisce quindi una versione maggiormente aggiornata del processo di quanto costituisca il attualmente il Manuale.

La Parte IV contiene la descrizione della modellazione e dei risultati: in particolare, il Cap. 9 contiene la descrizione dei meccanismi e degli aspetti salienti del modello, mentre il Cap. 10 descrive i problemi incontrati nell'applicare SLANG alla modellazione di un processo software industriale.

Il modello completo in linguaggio SLANG, comprensivo di brevi descrizioni esplicative che aiutano la comprensione del "codice" SLANG, è presentato nella Appendice A.

Infine, l'Appendice B contiene la sintassi "operativa" del linguaggio (la definizione ufficiale è oggetto di un lavoro parallelo a questo [Lip93]) impiegata per la stesura del modello, mentre l'Appendice C è una concisa introduzione al formalismo di reti di Petri di alto livello che costituisce il nucleo del formalismo SLANG.

Parte I

Il problema della gestione del processo software

Capitolo 2

Le idee e gli approcci nella gestione del processo software

In questo capitolo vengono approfondite le problematiche generali connesse alla gestione del processo di sviluppo del software, indicandone le tematiche principali all'interno dell'attività di ricerca nonché alcuni approcci ritenuti particolarmente significativi.

2.1 Concetti e definizioni generali

È opportuno innanzitutto definire l'oggetto del nostro studio, e cioè il *processo software*: secondo W.S. Humphrey [Hum89]

> *"The software process is the set of tools, methods, and practices we use to produce a software product"* [1].

I processi di produzione sono analizzati in ogni disciplina il cui obiettivo sia costruire prodotti: l'obiettivo di tali processi è rendere la produzione affidabile, predicibile ed efficiente. Tuttavia, va puntualizzato che il software possiede caratteristiche peculiari che ne rendono particolare la definizione del processo di produzione: in primo luogo, la produzione di software è un'attività in gran parte *intellettuale*, pertanto non completamente automatizzabile, in secondo luogo il software è caratterizzato da un'alta *instabilità*,

[1] "Il processo software è l'insieme di strumenti, metodi e pratiche impiegate per produrre un prodotto software"

ad esempio in termini di mutevolezza dei requisiti e conseguenti modifiche i prodotti.

A questo proposito, proprio l'instabilità del software genera uno dei problemi più importanti all'interno della gestione del processo, e cioè la *gestione del cambiamento* [2]. Questo può essere definito come segue [Mad92]:

- identificazione della necessità di produrre un cambiamento su un elemento dell'ambiente;

- acquisizione dell'adeguata conoscenza circa il cambiamento da produrre sull'elemento;

- valutazione dell'impatto del cambiamento su altri elementi dell'ambiente;

- selezione o costruzione di un metodo per l'esecuzione del processo di cambiamento;

- esecuzione dei cambiamenti;

- registrazione dei dettagli delle modifiche, al fine di garantirne la futura reperibilità;

- rilascio dell'elemento modificato all'interno dell'ambiente.

La gestione di un processo software, alla luce di queste considerazioni, impone la definizione di un qualche tipo di *modello* del processo; la sua assenza porta in genere ad una serie di effetti negativi, ad esempio:

- È difficile istituzionalizzare il processo in quanto le persone a cui è affidato tale compito possono trovarsi senza punti di riferimento.

- Gli sviluppatori, in mancanza di un modello comune, possono essere portati a compiere azioni scorrette, specie durante le fasi critiche del progetto.

- I manager hanno una cattiva *visibilità* del processo, cioè in altre parole è difficile comprendere il reale stato di avanzamento di un progetto e quindi le possibili azioni correttive.

[2] Spesso in letteratura ci si riferisce al processo di cambiamento del processo come al *metaprocesso*.

Tra le possibili ragioni per cui un'organizzazione può trovarsi senza un adeguato modello del processo ricordiamo le seguenti:

- Mancanza di comprensione del processo all'interno dell'organizzazione.
- La gestione dello sviluppo del prodotto avviene sempre in condizioni di "saturazione", cioè in situazioni di impiego totale di risorse con vincoli stringenti; in queste situazioni non vi è spazio per la definizione del processo.
- Riluttanza a definire i processi in quanto tali definizioni sono viste da parte di alcuni esperti come una riduzione del proprio ruolo chiave all'interno dell'organizzazione.
- L'organizzazione percepisce il valore di un prodotto come legato all'esecuzione del processo e non alla sua definizione, partendo dall'assunzione che in fondo i committenti pagano per i prodotti e non per i processi.

Per quanto riguarda l'ultimo punto, la situazione sta cambiando: come già accennato nel Cap. 1, aumenta sempre più il numero di standard relativi al processo di cui è richiesta una certificazione di conformità per poter ottenere una commessa, ragion per cui un numero sempre crescente di organizzazioni sta riconsiderando le convinzioni esposte prima alla luce di questi vincoli imposti dal mercato ma anche dei primi risultati prodotti dal lavoro di ricerca in questo settore.

Come già accennato all'inizio di questo lavoro, è possibile distinguere due aree di ricerca che si occupano del *process modeling* [3]: una che si occupa degli *aspetti manageriali* ed una che si occupa invece degli *aspetti tecnici*. Come esempio rappresentativo della prima area di ricerca, nella Sez. 2.2 illustreremo i concetti generali del Capability Maturity Model sviluppato al SEI, mentre nelle Sez. 2.3 e successive esamineremo i concetti generali e i principali approcci all'interno del process modeling.

[3] Nel seguito si preferirà utilizzare il termine *process modeling* anziché la traduzione italiana *modellazione di processo*.

2.2 L'approccio SEI: il Capability Maturity Model

2.2.1 Una breve storia

Nel novembre 1986 il Software Engineering Institute (SEI) iniziò lo sviluppo di un modello di maturità del processo software il cui scopo era guidare le organizzazioni nel miglioramento e nella valutazione di tale processo, il *Capability Maturity Model for Software (CMM)* [Hum89]. Tale sforzo venne intrapreso su richiesta del governo federale degli Stati Uniti, il quale necessitava di un metodo per valutare l'affidabilità dei fornitori di software. La prima versione del modello, stilata da W.S. Humphrey, venne distribuita nel 1987 insieme ad un questionario che avrebbe dovuto aiutare le aziende ad individuare le aree del processo in cui concentrare gli interventi.

In questi anni, forte di una sperimentazione sul campo, la SEI ha fatto evolvere il modello [PCC+91], fornendo insieme ad una versione più convincente di questo anche una serie di *"key practices"* cioè una lista dettagliata delle azioni tecniche e manageriali necessarie per percorrere dal basso verso l'alto i cinque livelli del modello.

Va puntualizzato che all'interno del mondo accademico non vi è uniformità di giudizi sul modello SEI: in particolare, le critiche più che accanirsi sul modello, si focalizzano proprio sulle *key practices*, considerate troppo prescrittive, in quanto forniscono un modello aprioristico e dogmatico senza margini di flessibilità che permettano di adattarlo facilmente alle peculiarità dei vari processi.

2.2.2 Il modello

In [PCC+91] si definiscono i seguenti concetti chiave:

Software process capability descrive i limiti dei risultati attesi dall'impiego di un determinato processo software. Di conseguenza, fornisce un mezzo per valutare i risultati più probabili derivanti dal prossimo progetto che l'organizzazione deciderà di intraprendere.

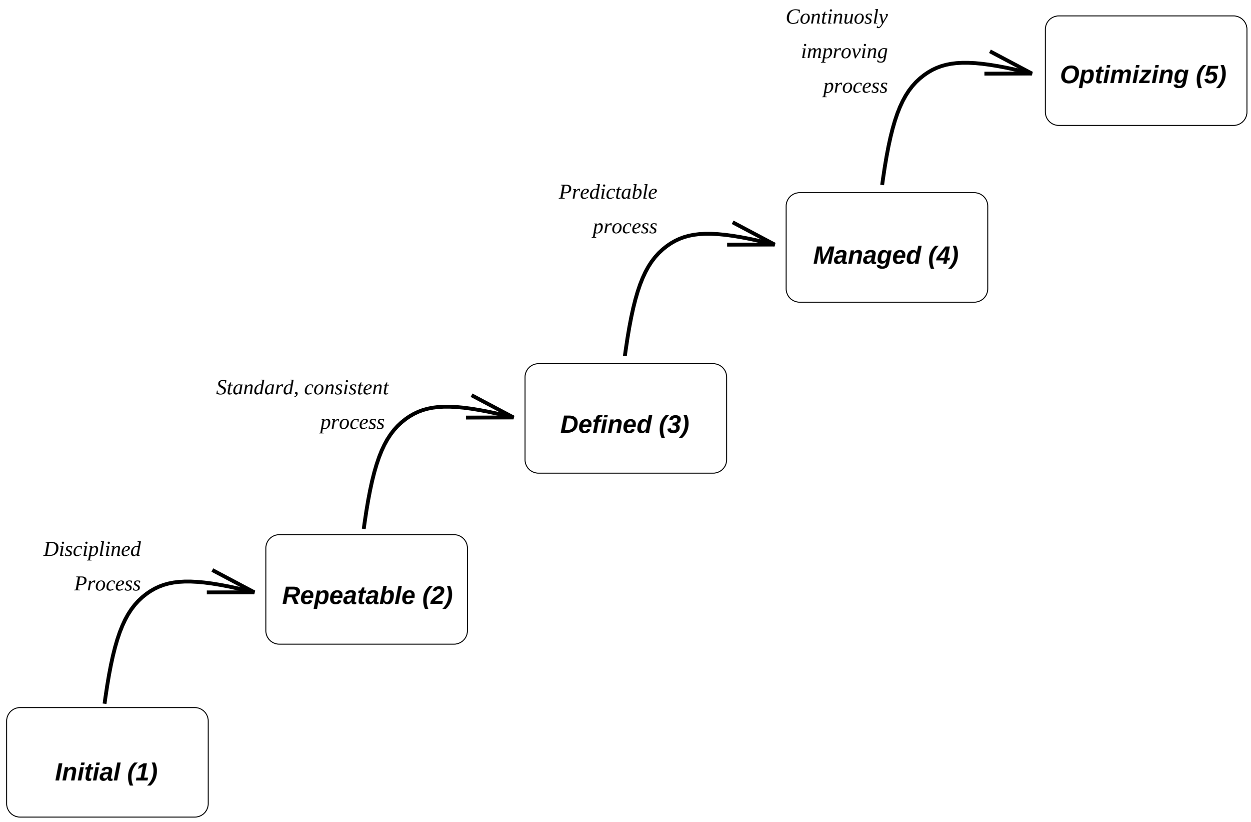

Figura 2.1: I livelli definiti nel Capability Maturity Model.

Software process performance rappresenta i risultati effettivamente ottenuti dall'impiego di un processo software. Si noti che a seconda del tipo di progetto e del contesto all'interno del quale esso è condotto, la performance di un progetto può non riflettere la piena *process capability* dell'organizzazione.

Software process maturity implica una crescita potenziale della *process capability*. Essa indica sia la bontà del processo software dell'organizzazione che la bontà della sua applicazione effettiva. Il CMM è basato sull'ipotesi che la *process maturity* sia un indicatore della *process capability*.

Il modello del Capability Maturity Model è strutturato in cinque livelli, mostrati in Fig. 2.1 (Da [PCC+91]):

Initial (1). Le organizzazioni non forniscono un ambiente stabile per lo sviluppo e la manutenzione del software. L'efficacia delle tecniche dell'Ingegneria del Software, quando impiegate, è sminuita dagli effetti di pianificazioni inefficaci e di commesse azzardate. Ciò è dovuto al fatto che i processi, durante le crisi, abbandonano le pianificazioni ed il processo stabilito per tornare ad un modello di sviluppo di tipo *code and fix*. Il processo software viene caratterizzato *ad hoc* e spesso in maniera caotica. Il successo di un progetto dipende unicamente dalle capacità dei singoli.

La *process capability* delle organizzazioni a Livello 1 è impredicibile poichè il processo è costantemente cambiato o modificato mentre il progetto viene sviluppato.

Repeatable (2). In questo livello, le politiche di gestione del progetto sono stabilite dall'organizzazione, mentre la pianificazione e la gestione di un nuovo progetto è basata sulle esperienze maturate in progetti analoghi. L'obiettivo di queste organizzazioni è il raggiungimento di una stabilità del processo. Le commesse sono scelte sulla base di esperienze precedenti. Sono definiti standard di progetto e si assicura la loro corretta applicazione.

Defined (3). In questo livello il processo standard viene documentato, sia per quanto riguarda lo sviluppo del software che per quanto riguarda la sua gestione. Le tecniche dell'Ingegneria del Software sono applicate e viene istituito un gruppo all'interno dell'azienda (Software Engineering Process Group, SEPG) che si occupa della definizione e del miglioramento del processo software. Vengono istituiti opportuni corsi di formazione per garantire che ogni attore del processo possegga le appropriate conoscenze sul medesimo.

Managed (4). A questo livello le organizzazioni sono in grado di fissare obiettivi *quantitativi* per i prodotti software: la produttività e la qualità vengono misurate e viene creata una base dati a livello di organizzazione per confrontare i dati raccolti dai vari progetti. A questo punto l'organizzazione dovrebbe essere in grado di distinguere i legami causa-effetto che generano variazioni nel processo, distinguendo queste ultime da quelle generate da variazioni casuali. I rischi derivanti dall'impiego di nuove tecnologie sono noti e gestiti attentamente.

Optimizing (5). Le organizzazioni a questo livello sono tese ad un continuo miglioramento del loro processo. Appositi gruppi esaminano la performance del processo alla ricerca di difetti: i processi sono valutati per prevenire il ricorrere di errori noti e l'esperienza è riversata nei successivi progetti. Le organizzazioni a Livello 5 cercano continuamente di innalzare il limite superiore della loro *process capability*: i miglioramenti avvengono sia per progressi incrementali che per innovazioni tecnologiche.

È interessante capire come varia la "visibilità" del processo così come percepita dal management: un diagramma esplicativo è mostrato in Fig. 2.2 (Da [PCC+91]).

Al Livello 1 il processo software è un'entità amorfa e la visibilità all'interno del processo è difficoltosa. Poiché la suddivisione delle attività è ambigua i managers si trovano in difficoltà quando si tratta di stabilire lo stato del processo e delle attività.

Al Livello 2 la definizione delle procedure base di gestione del processo fa sì che il processo diventi visibile solo nei punti prestabiliti. Il processo di sviluppo del prodotto diventa una sequenza di *black box*: anche se il management non conosce l'interno di tali attività è possibile stabilire dei punti di controllo.

Al Livello 3 la struttura interna delle fasi è visibile, perciò managers e tecnici conoscono il loro ruolo e le loro responsabilità all'interno del processo e come le loro attività interagiscono con esso.

Al Livello 4 il processo software è misurato quantitativamente, pertanto i managers sono in grado di capire progressi e problemi. La loro abilità di prevedere i risultati aumenta man mano che la variabilità del processo diminuisce.

Al Livello 5 il processo può e deve essere modificato, per adattarlo alle ottimizzazioni. I managers sono in grado di prevedere l'impatto e l'efficacia del cambiamento.

Il modello CMM non si limita tuttavia a fornire una semplice indicazione di livelli di maturità per il processo software, ma per ogni livello è definita una struttura che definisce gli aspetti operativi legati al modello, cioè in altre parole come organizzare un processo in modo da evolvere dal livello considerato verso livelli superiori. La struttura "nascosta" sotto un livello del modello CMM è presentata in Fig. 2.3 (Da [PCC+91]).

Ogni *Maturity Level* indica la *Process Capability* associata al livello con-

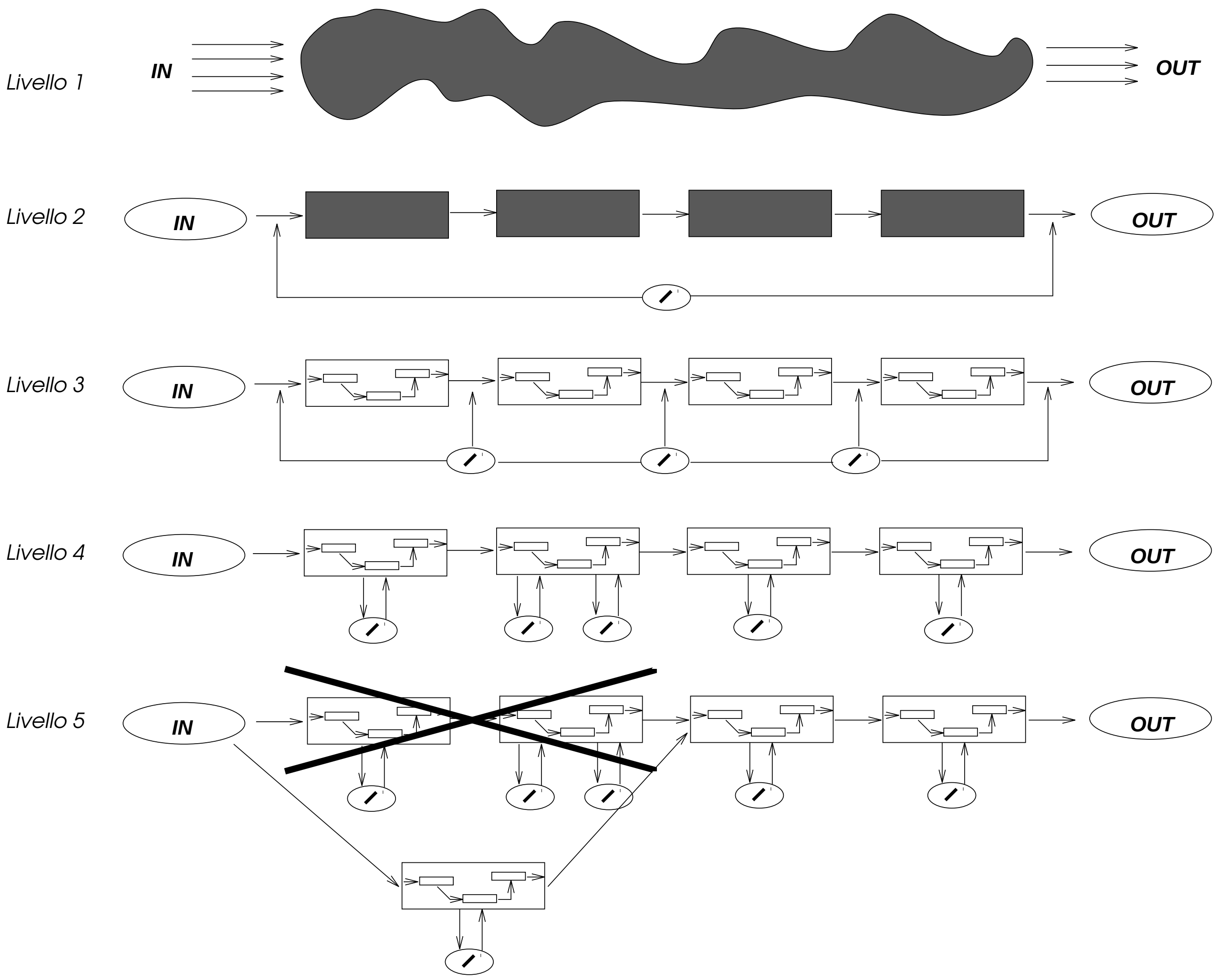

Figura 2.2: La visibilità del processo.

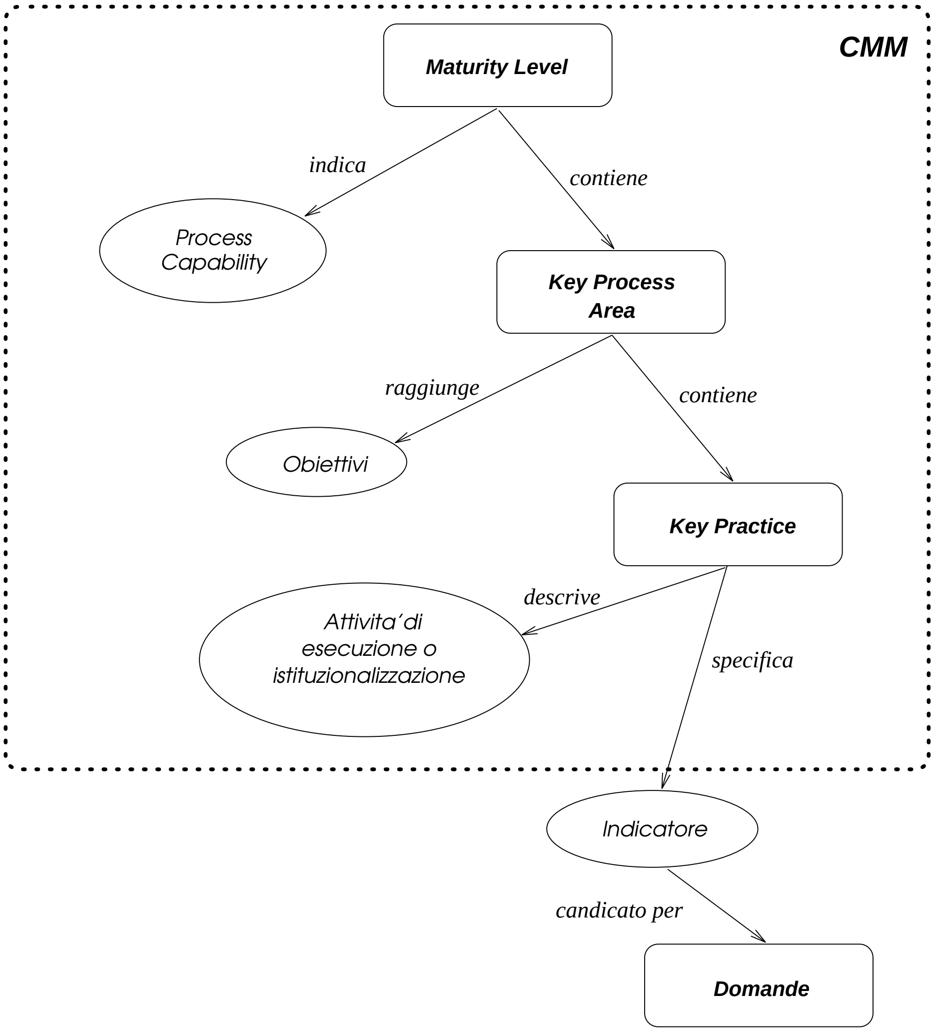

Figura 2.3: La struttura associata ad ogni livello del CMM.

siderato, e contiene un insieme di *key process area*, cioè di aree del processo che vanno create oppure tenute sotto controllo per garantire la possibilità di miglioramento del processo. Ad esempio per il Livello 2:

- Gestione della configurazione.
- Software Quality Assurance (SQA).
- Gestione delle commesse a terze parti.
- Pianificazione del progetto.
- Gestione dei requisiti.
- Monitoraggio e visibilità del progetto.

Ogni area così indicata contiene al suo interno una serie di *key practices*, cioè indicazioni *prescrittive*, peraltro assai dettagliate, delle azioni da intraprendere per guidare il miglioramento del processo. Lo scopo di tali indicazioni è da un lato fornire una guida per le attività di esecuzione e di istituzionalizzazione del processo, abbracciando quindi sia la gestione dello sviluppo che quella del management, dall'altro fornire una serie di indicatori che, attraverso la compilazioni di appositi questionari forniti dalla SEI, permettono l'identificazione delle aree carenti all'interno dell'organizzazione.

Contrariamente a quanto si potrebbe pensare, va puntualizzato che secondo la SEI ci sono rari esempi di *progetti* classificabili ai Livelli 4 e 5 ma nessun *processo* classificabile a tali livelli. La stragrande maggioranza delle aziende si trova ai Livelli 1 e 2, almeno per quanto riguarda le organizzazioni statunitensi, sulle quali la SEI ha condotto le indagini.

Una critica che può essere mossa al modello proposto dalla SEI è quello di essere troppo rigido: al di là della parte che riguarda le *key practices*, di cui abbiamo già commentato l'eccessivo livello di prescrizione, tutto il modello si presenta in maniera sufficientemente "dogmatica", presentando un modello plausibile ma forse con pochi margini di adattabilità a situazioni non convenzionali.

Può essere interessante a questo proposito citare un lavoro della SEI [HKG91] che ha preso in esame un insieme di organizzazioni giapponesi, cercando pertanto un confronto con il tradizionale antagonista tecnologico degli Stati Uniti. Aldilà delle varie imprecisioni ed approssimazioni presenti nell'articolo, è interessante rilevare che i dati mostrano che la stragrande maggioranza delle aziende giapponesi si collocano al Livello 1, anzi la maggior parte è spostata verso un ipotetico livello zero piuttosto che verso il passaggio al Livello 2; tra di esse si trovano inoltre un piccolissimo numero di aziende che posseggono un processo classificabile al Livello 4 e addirittura 5. Ciononostante il livello del prodotto delle aziende giapponesi a Livello 1 è assai più elevato delle corrispondenti aziende statunitensi.

È chiaro che questa situazione può essere spiegata facendo riferimento ad aspetti non strettamente tecnici, ma piuttosto socio-culturali, in quanto la cultura giapponese è incentrata sull'individuo e sulla sua capacità di dare tutto sé stesso per la realizzazione di un compito; è altrettanto chiaro quindi come il CMM non sia in realtà *il* modello, ma rappresenti soltanto uno dei possibili modelli del processo software.

A titolo di esempio, esiste almeno un'altro modello assai noto in letteratura [BR88] il quale parte da presupposti totalmente diversi dal CMM, impostando l'intera attività di gestione del processo sulla misurazione della qualità del prodotto e del processo.

2.3 Il process modeling

Il process modeling è un'area di ricerca relativamente nuova, pertanto non c'è un consenso generale circa i suoi obiettivi. Una lista possibile è la seguente [ABGM92]:

- **Comunicazione.** Il processo di sviluppo del software può essere visto come una struttura all'interno della quale i vari attori del processo lavorano in maniera cooperativa: essi ricoprono ruoli diversi, i quali dovrebbero essere ben chiari. Tuttavia, il processo è spesso lasciato implicito, pertanto non vi è un riferimento comune per il confronto tra le risorse umane impiegate in esso.

- **Previsione e pianificazione.** Da un progetto di tipo ingegneristico ci si aspetta un prodotto affidabile costruito in un tempo limitato mediante risorse altrettanto limitate. Nelle fasi iniziali del progetto, è compito del manager stimare le risorse necessarie, in termini di mesi/uomo e durata complessiva. La rappresentazione esplicita del processo fornisce una struttura rigorosa su cui compiere delle analisi, magari attraverso simulazioni ed animazioni.

- **Gestione e ripianificazione.** Il cambiamento è un aspetto critico all'interno del processo software, di cui costituisce un aspetto inscindibile, come evidenziato nella Sez. 2.1: il modello costituisce un valido punto di riferimento per individuare i punti esatti in cui operare il cambiamento e possibilmente simularne ed analizzarne gli effetti.

- **Misura.** Le misure forniscono una valutazione quantitativa del processo e della qualità del prodotto. Senza un riferimento comune, tuttavia, si rischia di rilevare grandi quantità di dati senza la possibilità di poterli interpretare in maniera coerente. Addirittura, la misurazione può essere direttamente inserita all'interno della descrizione del processo.

- **Configurazione.** Un processo software necessita di essere adattato alle esigenze specifiche del progetto a cui è applicato: inoltre, deve poter evolvere per seguire le necessità di miglioramento degli aspetti di qualità. Questo è un aspetto diverso da quello del cambiamento, che invece avviene *durante* l'esecuzione del processo. La definizione del processo specifica chi, come e con che strumenti deve essere eseguito il processo, fornendo dunque una specifica anche della configurazione.

- **Riuso.** La natura dei processi software non è tale da richiedere una loro ridefinizione da zero ogni volta che si utilizzano per un progetto anzi, dovrebbero essere impiegati per evidenziare le possibilità di riuso in termini di conoscenza sul processo: attualmente, invece, la forma di riuso più impiegata è il riutilizzo del personale.

- **Esecuzione.** La descrizione del processo definisce le attività eseguite all'interno di esso, sia da parte del management che dei tecnici. Se la notazione impiegata per la descrizione è eseguibile, quest'ultima può essere utilizzata per guidare il lavoro degli attori del processo attraverso un ambiente software: ciò è tra l'altro l'obiettivo del progetto SPADE.

- **Verifica.** L'uso di notazioni formali per la specifica del processo fornisce varie possibilità per la verifica del processo: ad esempio, una notazione basata su automi a stati finiti permette di verificare condizioni di raggiungibilità per i vari stati, analisi di assenza di *deadlock* e così via. Altra forma di verifica può essere fornita dalla simulazione, con la possibilità di eseguire analisi di tipo "*what if*".

È utile seguire una distinzione tra i tipi di modelli definibili, come proposto in [CKO92]:

- *Modelli prescrittivi:* impongono come il processo deve essere effettivamente eseguito. L'analisi di questo tipo di processi deve stabilire la capacità degli attori del processo di eseguirne le attività prescritte.

- *Modelli descrittivi:* cercano di descrivere il processo effettivamente utilizzato all'interno dell'organizzazione, ad esempio attraverso l'utilizzo di metriche.

- *Modelli proscrittivi:* delineano i comportamenti vietati all'interno del processo, definendo pertanto i vincoli a cui le operazioni svolte dagli attori devono sottostare.

I primi modelli del processo software sono da alcuni ([GJM91], [CKO92]) identificati con i cosiddetti *cicli di vita del software*, il cui capostipite è il ben noto *waterfall model* [Roy70], che ha costituito per anni un punto di riferimento nell'organizzazione del processo software. Questo tipo di rappresentazione, anche nelle proposte più recenti [Boe88], mantiene comunque il livello della descrizione ad un livello di astrazione troppo elevato per consentire una effettiva applicazione nella guida del processo produttivo.

La necessità di una descrizione maggiormente dettagliata del processo ha generato la necessità di appropriati formalismi da impiegare in tale compito: la seguente sezione illustra alcuni degli approcci presenti in letteratura.

2.4 Formalismi per il process modeling

Al fine di fornire effettivo supporto alle attività di modellazione, analisi ed automazione del processo, un *linguaggio per la specifica del processo software* dovrebbe possedere le seguenti caratteristiche [BFG93a]:

- Deve essere un linguaggio *formale*, in modo da permettere analisi ed esecuzione (*enaction* [4]) automatiche.
- Deve permettere al modellatore del processo di descrivere sia le attività che costituiscono il processo sia i risultati prodotti durante la loro esecuzione.
- I processi software sono sistemi *human-oriented*, cioè sistemi in cui gli esseri umani cooperano con i tool computerizzati al fine di ottenere un unico obiettivo; il formalismo dovrebbe fornire i mezzi necessari a definire tale interazione.
- L'architettura necessaria all'esecuzione del processo deve essere costituita da un ambiente eterogeneo, distribuito e concorrente, che fornisca supporto alla cooperazione ed all'interazione tra svariati agenti umani.
- I modelli di processo sono spesso assai estesi, ragion per cui sono *obbligatori* costrutti che permettano di impiegare costrutti di modellazione *in-the-large*, quali astrazione, *information hiding* e incapsulazione.
- Il linguaggio dovrebbe permettere vari tipi di analisi, sia statiche che dinamiche, attraverso la dimostrazione di proprietà formali e attraverso simulazioni ed animazioni.
- Il linguaggio dovrebbe permettere l'evoluzione del modello in risposta all'evoluzione del processo, per gestire il problema del cambiamento delineato in Sez. 2.1.

L'insieme (numeroso) di proposte in letteratura copre con varia densità l'insieme di requisiti ora delineati. Per permettere un confronto tra i vari formalismi proposti, è stato proposto nel 1990 [K$^+$90], all'interno della sesta edizione dell'International Software Process Workshop un esempio di processo sulla cui specifica confrontare i vari formalismi. I risultati di tale indagine sono stati proposti in [KR90] e hanno condotto ad una classificazione dei linguaggi sotto vari punti di vista, quali le funzionalità offerte, il linguaggio base, i meccanismi di esecuzione e così via.

[4] Tale termine viene correntemente utilizzato in letteratura a significare che l'esecuzione viene svolta in parte da tool ed in parte da esseri umani.

Seguendo la classificazione proposta nel Cap. 1, in cui si erano distinti gli approcci principali in approcci basati su linguaggi imperativi, sistemi basati sulla conoscenza e sistemi derivati dalla specifica di sistemi reattivi, vengono ora descritti i formalismi ritenuti maggiormente significativi all'interno di ciascuna di queste tre aree.

2.4.1 APPL/A

Il linguaggio APPL/A [TSY+88],[Ost87] è stato forse il prototipo dei linguaggi per la modellazione di processo. Esso è di fatto un linguaggio di programmazione di processo *imperativo* basato sul linguaggio Ada, di cui estende le funzionalità mediante un insieme di costrutti in grado di definire relazioni tra oggetti.

Essendo basato sul linguaggio Ada, APPL/A ne eredita le caratteristiche di base, fornendo dunque costrutti di modularizzazione, tipi e meccanismi di concorrenza: inoltre, la descrizione del processo è fornita in maniera procedurale, sebbene sia possibile descrivere alcuni aspetti, per esempio vincoli di consistenza, mediante regole e quindi caratterizzando questo linguaggio come un approccio *multiparadigma*.

Tuttavia, non vi è la possibilità di descrivere il *metaprocesso*, cioè le politiche di evoluzione del processo.

2.4.2 Marvel

Marvel [KFP88],[BK91] è uno dei pochi sistemi effettivamente disponibile e funzionante all'interno del panorama degli ambienti per la gestione del processo software basati su linguaggi di modellazione.

L'approccio seguito è quello a regole, tipico dell'Intelligenza Artificiale; in particolare, le regole sono suddivise in tre insiemi:

- *Regole di progetto:* definiscono gli aspetti specifici del processo. Sono a loro volta distinte in:
  - *Regole di attivazione:* controllano l'inizio delle attività di sviluppo, e tipicamente coinvolgono l'attivazione di un tool: in genere hanno un insieme di effetti finali mutuamente esclusivi.
  - *Regole di inferenza:* definiscono relazioni tra gli attributi degli oggetti.

- *Regole di tipo:* definiscono i dati mediante tipi di dato orientati agli oggetti.
- *Regole di tool:* descrivono le interfacce con i tool esterni impiegati per lo sviluppo durante il processo.

Le regole possono essere applicate sia secondo il *forward chaining* sia mediante *backward chaining*, in maniera analoga ad altri approcci che impiegano regole, quali Merlin [HJP+90] o Articulator [MS90]: il forward chaining è impiegato quando è necessario guidare l'attore del processo in una sequenza prescritta di passi, mentre il backward chaining viene impiegato per identificare le alternative tra varie operazioni da proporre all'esecutore.

Marvel fornisce un supporto soltanto parziale alle problematiche del metaprocesso.

Uno degli svantaggi evidenziati all'interno di questo approccio e dell'approccio a regole in generale è l'estrema difficoltà che si incontra nel modificare le regole all'interno di una specifica complessa, dato l'alto numero di *effetti collaterali* che si vengono così a produrre.

2.4.3 Entity Process Models

Gli Entity Process Models (EPM, [Kel89],[KH89b]) sono una notazione basata su STATEMATE, un prodotto commerciale nato per la specifica di sistemi in tempo reale.

Sono distinti tre tipi di diagrammi a descrivere tre "viste" ortogonali sul processo:

- *Vista funzionale:* rappresenta *quali* compiti sono eseguiti e *quali* flussi di informazione sono pertinenti a tali compiti.
- *Vista comportamentale:* rappresenta *quando* i compiti sono eseguiti e *come*, cioè attraverso cicli, retroazioni, interazioni e così via.
- *Vista organizzativa:* specifica *dove* e *da chi* all'interno dell'organizzazione i compiti sono svolti, e con quali meccanismi fisici di comunicazione.

Il punto forte di questo approccio è la possibilità di sfruttare il tool STATEMATE per eseguire un gran numero di analisi, sia di tipo statico, nella costruzione dei diagrammi o nella verifica di raggiungibilità degli stati, che di tipo dinamico, eseguendo simulazioni in grado di verificare ad esempio la validità delle pianificazioni e dei criteri di allocazione.

Questo tipo di approccio è stato impiegato per la descrizione di un complesso sistema reale, all'interno del Dipartimento della Difesa degli Stati Uniti. Tale caso di studio, presentato in [KH89a], ha comportato la modellazione, attraverso il formalismo sommariamente descritto precedentemente, del processo di modifica della documentazione tecnica relativa al software di gestione degli aviogetti F-16 quindi, ancora una volta un processo di cambiamento ma di tipo diverso da quello da noi modellato. Il test sul formalismo risulta essere comunque abbastanza convincente, sebbene scarsamente documentato nell'articolo citato, in quanto la scala del processo preso in esame era assai elevata e in un processo altamente critico.

Parte II

Il linguaggio del modello

Capitolo 3

Il linguaggio SLANG per la modellazione di processo

Il linguaggio utilizzato per la modellazione del caso di studio reale oggetto di questo lavoro è SLANG (Spade LANGuage), attorno al quale è costruito l'ambiente per la gestione del processo software SPADE (Software Process Analysis, Design and Enactment), attualmente in sviluppo presso il CEFRIEL ed il Politecnico di Milano.

SLANG ([BFG91], [BFGG92], [BFG93b], [BF93], [BFG93a])è basato su un'estensione ad alto livello delle reti di Petri, le reti Ambienti/Relazioni (*Environment/Relationship Nets* nel seguito reti ER), la cui definizione formale è presentata in [GMMP91]. L'obiettivo delle reti ER è fornire un nucleo costituito da un formalismo che aumenti l'espressività delle reti di Petri, ma ne mantenga il rigore formale, consentendo la creazione di linguaggi di livello ancora più alto costruiti su questo nucleo definito formalmente.

Questo approccio, alla base della filosofia delle reti ER, è stato messo in opera per la definizione di SLANG il quale, pur estendendo tale formalismo con costrutti di più alto livello, possiede una semantica che può essere definita formalmente e completamente in termini di reti ER.

SLANG è ormai un linguaggio maturo, tanto da poter essere utilizzato per un compito di modellazione come quello oggetto di questo lavoro, ma necessita ancora di raffinamenti e modifiche per poter adattare meglio le sue caratteristiche ad un utilizzo reale nelle problematiche sollevate dalla modellazione: obiettivo di questo lavoro è stato anche il segnalare, e possibilmente proporre soluzioni per quelle caratteristiche di SLANG totalmente o parzialmente inadatte o mancanti. I risultati di questo particolare aspetto del lavoro verranno presentati nel Cap. 10.

In questo capitolo esamineremo completamente le caratteristiche del linguaggio, soffermandoci in maggiore dettaglio su quelle utilizzate all'interno della modellazione. Si considereranno noti i concetti base delle reti di Petri ed in particolare delle reti ER. Per quanto riguarda le reti di Petri, si può consultare [FGM+91] oppure [GJM91], mentre una breve introduzione alle reti ER è riportata nell'Appendice C.

3.1 Introduzione a SLANG

Il linguaggio SLANG è stato progettato sulla base dei requisiti per un linguaggio di modellazione di processo descritti nel Cap. 2. Inoltre, la scelta del formalismo delle reti di Petri come base per la progettazione di tale linguaggio risponde ad un'analogia tra il campo dei processi di sviluppo del software ed i sistemi reattivi proposta da Balzer [Bal90] e che può essere così riassunta:

- Il sistema interagisce con attività esterne che possono generare *eventi* in istanti impredicibili.
- Il sistema è composto di un insieme di task *concorrenti* che devono essere schedulati al fine di ottenere una scadenza globale.
- La correttezza globale dipende da *vincoli temporali*.

Sebbene questo tipo di analogia non sia in realtà così stretta, in quanto si è mostrato ad esempio che la natura dei vincoli temporali nei due sistemi è spesso profondamente diversa, pone comunque in luce alcuni aspetti rilevanti: in primo luogo è centrale il concetto di evento asincrono, generato all'interno del processo software e che da quest'ultimo deve essere gestito. In secondo luogo, le attività svolte all'interno del processo sono concorrenti e non indipendenti, pertanto assumono importanza cruciale le problematiche di gestione della concorrenza al fine di limitare i conflitti massimizzando e regolando il parallelismo. Infine, è evidenziata l'importanza basilare della gestione degli aspetti temporali, tanto maggiormente accresciuta dal fatto che spesso, nella pratica industriale, una cattiva gestione di questi aspetti porta a perdite assai elevate in termini economici.

Il linguaggio SLANG offre costrutti per affrontare questi ed altri problemi: in particolare:

- Gli eventi sono direttamente rappresentati dalle reti di Petri attraverso la nozione di transizione, mentre l'estensione a tale concetto presente nelle reti ER permette anche di specificare precondizioni e postcondizioni per essi.
- Esiste una nozione distribuita di *stato del processo*, direttamente ereditata dal formalismo delle reti di Petri.
- È fornito un costrutto di *attività* che viene utilizzato per permettere la modellazione in modo intuitivo delle attività concorrenti e cooperanti del processo software.
- Il processo guida l'esecuzione dei compiti da parte degli *esecutori* del processo: la comunicazione tra processo ed esecutore (essere umano o strumento di sviluppo) è assicurata mediante appositi costrutti, in particolare le *black transition* ed i posti di interfaccia utente (*user interface place*).
- Sono forniti costrutti per la modellazione dei vincoli temporali: in particolare è possibile specificare vincoli sugli eventi mediante i meccanismi direttamente ereditati dalle reti ER, oppure su sequenze complesse di eventi, mediante il costrutto di *timeout*.
- I dati del processo sono rappresentati mediante un paradigma orientato agli oggetti, pertanto è possibile associare loro dei metodi per specificare le operazioni permesse su di essi. Il linguaggio supporta le caratteristiche dell'approccio orientato agli oggetti, quali ereditarietà, late binding, polimorfismo.
- Il linguaggio fornisce caratteristiche per la modellazione *in-the-large* del processo, utilizzando concetti ereditati dai linguaggi di programmazione, quali modularizzazione, information hiding, incapsulazione.

In aggiunta a queste caratteristiche, SLANG permette di descrivere non solo l'esecuzione ma anche l'*evoluzione* del processo: in altre parole il linguaggio offre caratteristiche riflessive in grado di descrivere il *metaprocesso*, cioè i cambiamenti apportati attraverso il processo al processo medesimo.

Nelle successive sezioni verranno esaminate in maggior dettaglio le caratteristiche del linguaggio. La Sez. 3.2 descrive gli aspetti basilari del linguaggio, mentre le Sez. 3.3 e 3.4 ne affrontano gli aspetti più originali (e complessi).

3.2 Concetti e costrutti base del linguaggio

Un modello del processo software scritto in SLANG può essere definito matematicamente come una coppia di insiemi:

$$SLANGModel = \{ProcessTypes, ProcessActivities\}$$

dove *ProcessTypes* è un insieme di definizioni di tipi organizzate in una gerarchia basata sulla relazione di ereditarietà mutuata dall'approccio orientato agli oggetti, mentre *ProcessActivities* è un'insieme di definizioni di attività. Una definizione di attività in SLANG è sostanzialmente una rete di Petri di alto livello dove posti, archi e transizioni sono stati arricchiti con ulteriori informazioni, come descritto nel seguito.

3.2.1 I tipi

Durante un processo software vengono prodotte, usate e manipolate grandi quantità di dati di varia natura, dai documenti di specifica al codice eseguibile. In SLANG tutti i dati del processo sono tipizzati mediante tipi definiti attraverso un approccio orientato agli oggetti: in altre parole, i tipi sono organizzati in nella gerarchia `ProcessTypes` definita mediante la relazione *is_a* (oppure *subtype_of*). Secondo tale relazione, un sottotipo T_S di T è uno qualsiasi dei discendenti diretti o indiretti di T nella gerarchia, incluso T. La Fig. 3.1 illustra il frammento predefinito di tale gerarchia, la cui radice è il tipo `ProcessData`. I tipi ereditano attributi e operazioni dai supertipi.

Anche le attività SLANG, la cui definizione è contenuta nell'insieme ProcessActivities visto precedentemente, sono rappresentate come dati del processo, in maniera da poter fornire le caratteristiche riflessive che verranno discusse nella Sez. 3.4. In particolare, esistono quattro tipi predefiniti, `Token`,

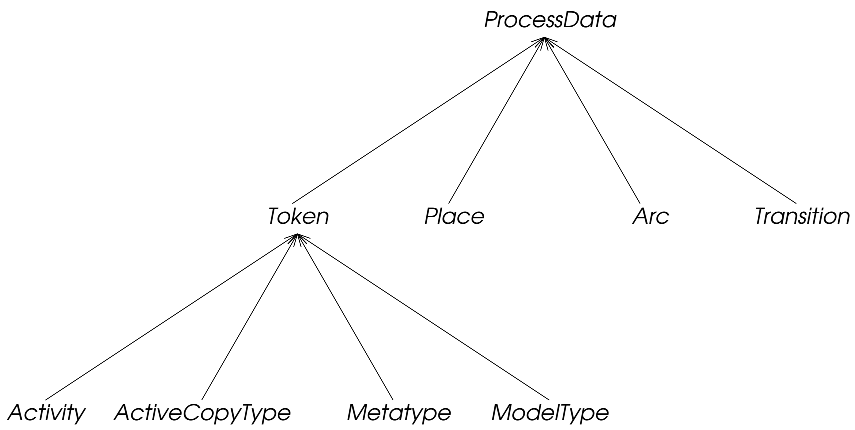

Figura 3.1: La parte predefinita della gerarchia ProcessTypes.

`Place`, `Arc` e `Transition` usati per implementare i rispettivi costrutti del linguaggio. Inoltre, sono forniti altri tre tipi predefiniti:

- `Activity`. Le istanze di questo tipo sono definizioni di attività.
- `MetaType`. Le istanze di questo tipo sono definizioni di tipo.
- `ActiveCopy`. Le istanze di questo tipo sono istanze in esecuzione, come risulterà più chiaro nella Sez. 3.4.

Questi tipi sono predefiniti e non modificabili da parte di chi realizza una specifica SLANG. I tipi definiti dall'utente (dove per utente si intende il modellatore del processo) sono contenuti nel sottoalbero che ha come radice un quarto tipo predefinito, il tipo `ModelType`.

La sintassi utilizzata per la definizione dei tipi è descritta nell'Appendice B.

3.2.2 Le attività

Come visto precedentemente, *ProcessActivities* è un insieme di definizioni di attività. In particolare, ogni definizione di attività è un'istanza del tipo `Activity` ed è descritta mediante una rete di Petri ad alto livello, quindi mediante un insieme di posti P, un insieme di transizioni T, ed un insieme di archi A.

Lo stato dell'attività è dato dalla marcatura della rete, cioè dall'assegnazione dei tokens ai posti, mentre gli eventi interni all'attività sono rappresentati dalle transizioni.

Il verificarsi di un evento, rappresentato dall scatto della transizione corrispondente, modifica lo stato dell'attività rimuovendo i tokens dai posti in ingresso alla transizione ed aggiungendo posti in uscita alla medesima. La topologia della rete descrive relazioni di precedenza, conflitti e parallelismi tra eventi ed attività. Ogni attività corrisponde ad un'unità logica di lavoro, la quale può includere eventi locali, invocazioni ad altre attività ed interazioni con l'ambiente, cioè con gli esecutori del processo (tool e risorse umane).

In accordo con i principi dell'information hiding, la definizione di un'attività è costituita da due parti, una di *interfaccia*, che permette l'interazione con il resto del modello, ed una di *implementazione* che rimane invece nascosta. L'interfaccia è composta da:

- un insieme di posti di interfaccia $P_{int} \subset P$;
- un insieme di transizioni di interfaccia $T_{int} \subset T$;
- un insieme di archi di interfaccia $A_{int} \subset A$, che connettono i posti di interfaccia con le rispettive transizioni.

Gli eventi che iniziano e terminano l'esecuzione di un'attività sono rappresentati mediante transizioni d'interfaccia, le quali sono partizionate in due insiemi:

- SE, insieme degli *eventi d'inizio*;
- EE, insieme degli *eventi di fine*.

tali che valga la seguente relazione:

$$(SE \cap EE = \emptyset) \wedge (SE \cup EE = T_{int})$$

Si dice che l'attività inizia (termina) non appena una qualsiasi delle transizioni in SE (EE) scatta. I posti di interfaccia in P_{int} sono divisi a loro volta in tre insiemi:

- *Posti di input:* sono posti di ingresso per qualcuno degli eventi d'inizio.
- *Posti di output:* sono posti di uscita per qualcuno degli eventi di fine.

- *Posti condivisi (shared places):* sono condivisi da attività diverse e possono essere posti d'ingresso o uscita per una qualsiasi delle transizioni nella parte di implementazione.

I posti d'interfaccia possono essere visti come "parametri formali" di un'attività: ogni altro posto interno alla parte di implementazione viene chiamato *posto locale*. La parte di implementazione descrive come deve essere eseguita l'attività mostrando le relazioni tra gli eventi rilevanti che possono accadere durante la sua esecuzione. Inoltre, l'implementazione di un'attività può contenere invocazioni ad altre attività, con un vincolo costituito dal fatto che *le invocazioni non possono essere ricorsive*.

L'invocazione di un'attività è rappresentata graficamente da un rettangolo con gli spigoli arrotondati (vedi Fig. 3.2): si noti che questa non è altro che una notazione sintattica per semplificare la scrittura della specifica in quanto, come sarà chiarito nella Sez. 3.4, la semantica di tale notazione può essere espressa come una rete che implementa la procedura di invocazione.

Poiché l'implementazione di un'attività può contenere invocazioni ad altre attività è possibile definire la relazione *usa* tra definizioni di attività, tale che A *usa* B se e soltanto se l'implementazione di A contiene un'invocazione a B. La relazione *usa* può essere rappresentata mediante un grafo, dove i nodi rappresentano le attività ed esiste un arco diretto da A a B se e solo se A *usa* B. Poiché le invocazioni ricorsive non sono permesse in SLANG, la relazione è una gerarchia, pertanto il grafo è in realtà un DAG (Direct Acyclic Graph). Tale grafo non viene modificato se il modello del processo non viene cambiato, pertanto tale grafo viene chiamato *DAG statico delle attività*. Si noti che in una specifica SLANG esiste solo una attività che non viene invocata da nessun'altra, la quale si trova alla radice del DAG: essa viene chiamata attività *radice (root activity)* e viene generata da un apposito programma di boot al fine di iniziare l'esecuzione del processo.

Vediamo ora gli oggetti che costituiscono l'implementazione di un'attività, rappresentati graficamente in Fig. 3.2.

Posti e token

Ogni posto ha un nome ed un tipo, e si comporta come un contenitore di token che può ricevere soltanto token del proprio tipo o di qualunque suo

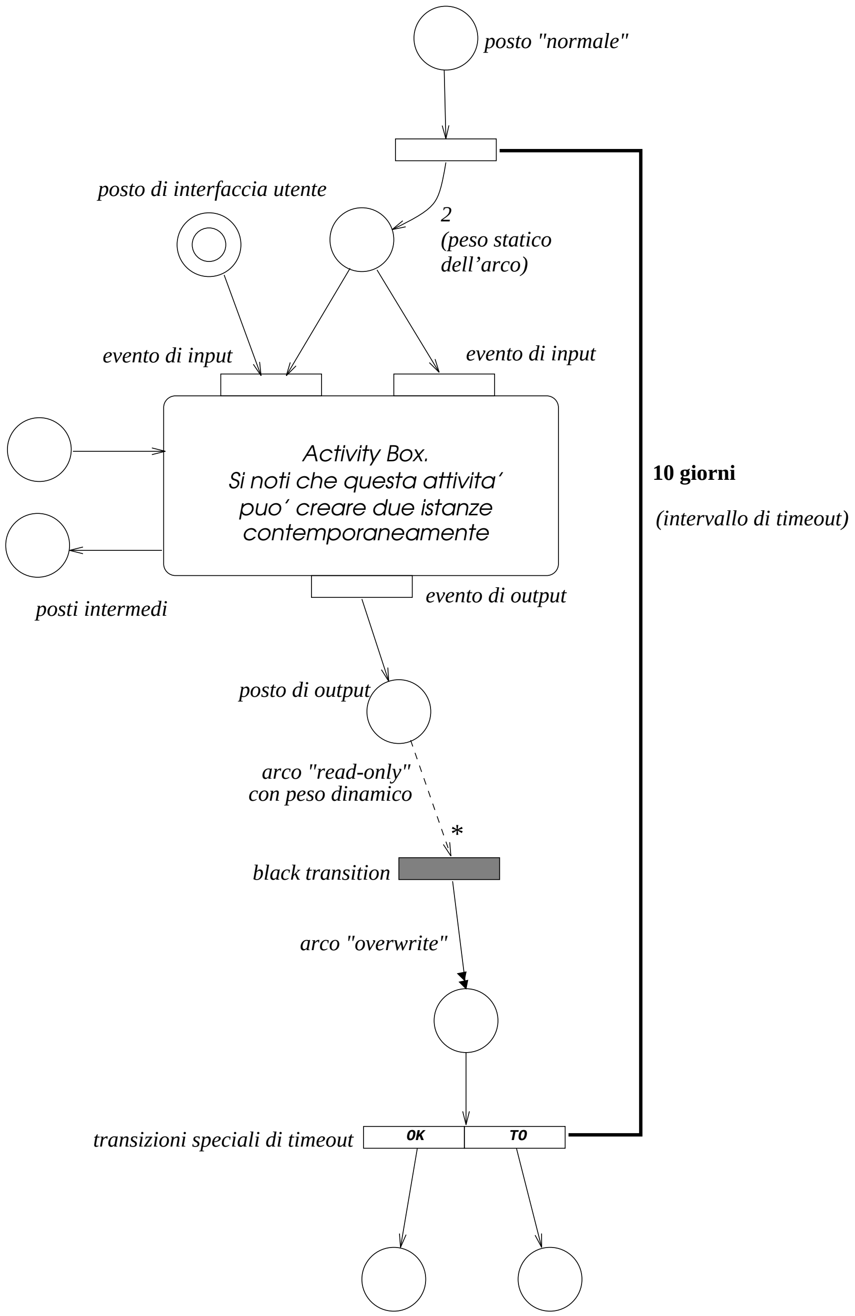

Figura 3.2: Un esempio fittizio di modello per mostrare la sintassi di SLANG.

sottotipo. Il contenuto di un posto può essere modificato dallo scatto di una transizione: la sola eccezione a questa regola sono i *posti di interfaccia utente*, i quali sono una classe speciale di posti che possono cambiare i loro contenuti in seguito all'intervento umano, attraverso l'interfaccia utente fornita dal sistema. I posti di interfaccia utente sono rappresentati mediante un doppio cerchio.

Per quanto detto precedentemente, dal momento che è possibile avere posti di tipo `Activity`, `Metatype` o `ActiveCopyType`, le definizioni delle attività e dei tipi, nonché gli stati delle attività possono essere manipolati come qualsiasi altro valore. Le implicazioni di questo fatto verranno discusse nella Sez. 3.4.

Transizioni

Le transizioni rappresentano eventi il cui verificarsi avviene per una durata temporale trascurabile, benché vincolabile all'interno di determinate finestre temporali, come risulterà chiaro nella Sez. 3.3.

Ad ogni transizione sono associate una *guardia* ed un'*azione*: la guardia è un predicato sui token che appartengono ai posti d'ingresso della transizione ed è usata per decidere se una tupla di tali token abilita o meno la transizione. Una tupla che soddisfa la guardia viene detta *tupla abilitante*.

Il comportamento dinamico di una transizione è determinato dalla *regola di scatto*, la quale afferma che quando una transizione scatta, i token che soddisfano la guardia sono rimossi dai posti in ingresso alla transizione e viene eseguita l'azione associata alla transizione. Come risultato dell'esecuzione dell'azione, una tupla di output è inserita nei posti d'uscita della transizione.

In SLANG esiste però un tipo speciale di transizione, utilizzato per modellare l'attivazione di tool software utilizzati all'interno del processo di sviluppo: la *black transition*. In essa, l'azione contiene una chiamata ad una routine eseguibile non-SLANG, quale ad esempio un eseguibile UNIX, la quale viene eseguita in maniera *asincrona*. Ciò significa che altre transizioni possono scattare mentre una black transition sta eseguendo la propria azione, ed inoltre significa anche che la black transition, contrariamente alle transizioni tradizionali, viene eseguita in un tempo tutt'altro che trascurabile (si pensi all'invocazione di un compilatore).

Archi

In SLANG agli archi sono associati dei pesi, il cui valore di default è 1. Il peso indica il numero di token che fluiscono attraverso ad ogni scatto della transizione. Tale peso può essere un numero definito staticamente oppure calcolato dinamicamente, nel qual caso esso viene indicato mediante un asterisco. In quest'ultimo caso, inoltre, il valore del peso è noto solo a run-time e può variare ad ogni occorrenza dello scatto. Ciò è utile per modellare situazioni in cui è necessario selezionare ad esempio *tutti* i token che soddisfano una determinata proprietà.

Oltre agli archi per così dire "tradizionali", SLANG offre anche altri due tipi di archi, i quali sono in realtà notazioni sintattiche che rendono più semplice la scrittura della rete ma non estendono la semantica del linguaggio. Tali archi, mostrati in Fig. 3.2, sono:

- *Archi read-only:* possono essere utilizzati per collegare un posto con una transizione. La transizione può leggere i valori dei token presenti nei posti d'ingresso al fine di valutare la guardia e l'azione, ma nessun token viene prelevato da essi durante lo scatto.

- *Archi overwrite:* possono essere utilizzati per collegare una transizione con un posto. Quando la transizione scatta, la seguente sequenza atomica di operazioni viene eseguita:

  - Il posto viene svuotato di tutti i token in esso contenuti.
  - I token prodotti dallo scatto vengono inseriti all'interno di tale posto.

  In questo modo, l'effetto globale è un'effettiva sovrascrittura del contenuto del posto.

3.3 Rappresentazione degli aspetti temporali

Come già accennato, gli aspetti temporali rivestono un ruolo cruciale all'interno della gestione del processo di sviluppo del software. In questa sezione verranno illustrati i meccanismi presenti nella versione attuale del linguaggio: il

Cap. 10 illustrerà come alcuni di tali meccanismi, in particolare il timeout, siano inadatti ad una efficace modellazione di un processo software reale. Successive versioni del linguaggio terranno conto di tali problemi.

In SLANG ogni token contiene implicitamente un *timestamp* che rappresenta l'istante temporale in cui il token è stato prodotto; inoltre, si assume che tutti i token prodotti da una medesima transizione posseggano il medesimo timestamp, corrispondente all'istante di scatto della transizione.

Durante la progettazione della rete, vengono associati dei *vincoli temporali* ad ogni transizione, ciascuno dei quali definisce implicitamente la finestra temporale all'interno della quale la transizione può essere abilitata a scattare. In particolare, possono essere definite le seguenti due semantiche temporali, analoghe a quelle caratteristiche delle reti ER:

- *Semantica temporale forte:* la transizione *deve* scattare entro l'intervallo temporale concesso, a meno che non venga disabilitata da un'altra transizione in conflitto.

- *Semantica temporale debole:* la transizione *può* scattare o meno entro l'intervallo temporale concesso. Se non scatta all'interno di tale intervallo, perde il diritto a scattare.

L'elenco dei vincoli temporali forniti da SLANG è riportato in Tab. 3.1: il loro significato è riportato in termini di intervalli e di tipo di semantica. Date due costanti α e β espresse in unità temporali, l'intervallo temporale $[\alpha, \beta]$ specifica che la transizione t è abilitata a scattare in un'istante compreso tra $T_e + \alpha$ e $T_e + \beta$, dove T_e è l'istante di abilitazione (cioé il massimo timestamp all'interno della tupla abilitante).

Il linguaggio SLANG permette anche di fissare vincoli temporali non solo su singoli eventi, ma anche su sequenze complesse di eventi, mediante il costrutto di *timeout*. Il timeout permette di stabilire un intervallo di tempo massimo che può trascorrere tra l'occorrenza di due eventi, i quali sono rappresentati nella rete da due transizioni, una transizione di *inizio timeout* ed una di *fine timeout*. In particolare, la transizione di fine timeout è composta in realtà di due transizioni, delle quali una rappresenta l'evento associato al completamento nel tempo indicato della sequenza di eventi, mentre l'altra rappresenta l'evento associato al mancato raggiungimento della scadenza.

Notazione	*Semantica*	*Intervallo*
ASAP	strong	$[0, \infty]$
ASAP AFTER α	strong	$[\alpha, \infty]$
MAY BEFORE α	weak	$[0, \alpha]$
AFTER α *AND MAY BEFORE* β	weak	$[\alpha, \beta]$

Tabella 3.1: La notazione temporale utilizzata in SLANG.

Un timeout è descritto attraverso un intervallo di tempo $\triangle$ ed un predicato $\prod$ che pone in relazione le tuple di ingresso delle transizioni di inizio timeout e fine timeout. Allora, *il timeout è verificato se, dopo lo scatto della transizione di inizio timeout con una tupla abilitante* t_{start}, *la transizione di fine timeout viene abilitata entro* $\triangle$ *da una tupla* t_{end} *tale che il predicato* $\prod(t_{start}, t_{end})$ *sia verificato.*

Si noti che la semantica del timeout può essere data in termini di reti ER, quindi tale costrutto equivale di fatto ad una *macro* costruita su tali reti.

3.4 Meccanismi di evoluzione del processo

Come già accennato precedentemente, in SLANG il modello del processo può includere non solo la descrizione del processo di sviluppo del software, ma anche quella del metaprocesso. Le caratteristiche riflessive del linguaggio consentono di manipolare il modello del processo ed il suo stato nella stessa maniera in cui ogni altro dato del processo è manipolato.

Questa sezione presenta gli aspetti i meccanismi di esecuzione (nel senso di *enaction*) di SLANG, con riferimento anche ai meccanismi che consentono l'evoluzione del modello del processo in quanto, sebbene tali aspetti non siano trattati all'interno della modellazione, tali meccanismi sono invece sfruttati da alcune delle attività della specifica, come sarà chiarito nel Cap. 9.

3.4.1 L'interprete SLANG

L'interprete del linguaggio SLANG è il responsabile dell'esecuzione dei modelli di processo scritti mediante tale formalismo. Quanto segue è una descrizione intuitiva del suo funzionamento.

Ogni volta che un'attività deve essere eseguita, viene creata una nuova istanza dell'interprete SLANG. Useremo il termine *process engine* per indicare una delle istanze in esecuzione dell'interprete.

L'attività radice di un modello SLANG viene eseguita da un process engine iniziale, il quale è generato da un apposito programma di boot. In ogni altro caso, i process engine sono creati usando black transitions nelle quali il tool invocato è l'interprete SLANG. Tali black transitions posseggono un solo posto di input ed uno di output, ambedue di tipo (predefinito) `ActiveCopyType`. Le istanze di tale tipo sono chiamate *copie attive* (active copies) di un'attività. Ogni copia attiva contiene tutta l'informazione necessaria all'esecuzione di un'attività, in particolare contiene la definizione dell'attività insieme alle definizioni di tipo usate dall'attività medesima, lo stato della copia attiva ed un riferimento agli eventi di inizio o fine insieme alle tuple abilitanti o di uscita, rispettivamente. Lo stato della copia attiva è definito dalla marcatura dei suoi posti.

Allorché un process engine inizia la propria esecuzione, esso riceve come ingresso una copia attiva: le definizioni dell'attività e dei tipi individuano il codice da eseguire, lo stato della copia attiva viene preso come stato di esecuzione iniziale e l'esecuzione inizia con lo scatto della transizione dell'evento iniziale data la corrispondente tupla abilitante. Tutto ciò vale se non vi sono invocazioni di attività all'interno di quella considerata: questo caso è preso in considerazione nella Sez. 3.4.2.

Iniziando dallo stato iniziale fornito dalla copia attiva, il process engine valuta le guardie associate con le transizioni per verificare se esiste qualche tupla di ingresso che abiliti una o più transizioni. In tal caso, ne viene scelta una e viene fatta scattare. Lo scatto rimuove i token dai posti di ingresso (o li legge nel caso di archi read-only), esegue l'azione corrispondente ed inserisce i token prodotti nei posti d'uscita (eventualmente sovrascrivendo i precedenti nel caso di archi overwrite).

Si noti che lo scatto di una transizione è un'evento atomico, pertanto non vi è alcuno stato intermedio dello scatto visibile ad altri eventuali process engine. Nel caso di una black transition, tuttavia, lo scatto corrisponde alla chiamata di un processo non-SLANG che viene eseguito in maniera asincrona: in tal caso i token sono rimossi dai posti in ingresso quando tale processo viene generato, mentre solo alla terminazione di tale processo i risultati sono copiati all'interno dei posti d'uscita della black transition.

Il verificarsi di un qualsiasi evento, cioè dello scatto di una qualsiasi transizione, produce un cambiamento dello stato della copia attiva che può abilitare o disabilitare altre transizioni. Un'attività termina allorché uno dei suoi eventi di fine scatta: a questo punto termina anche l'esecuzione dell'interprete e la copia attiva risultante viene prodotta nel posto di uscita della black transition che aveva generato il process engine. Tale copia attiva contiene la definizione dell'attività e dei suoi tipi, lo stato finale e l'evento di fine, insieme alla tupla di uscita prodotta dallo scatto.

3.4.2 L'invocazione di attività

Come già accennato in precedenza, sebbene l'invocazione di attività sia un costrutto SLANG, essa non aggiunge potere semantico al linguaggio, ma costituisce semplicemente una notazione sintattica più leggibile ed usabile: in altre parole, la semantica di un'invocazione SLANG può essere definita usando i costrutti di base forniti dal linguaggio, come mostrato in Fig. 3.3.

Da tale figura si nota che:

1. L'esecuzione dell'attività B inizia quando uno dei suoi eventi d'inizio è selezionato per lo scatto da parte del process engine di A, supposto che A sia l'attività che invoca B.

2. L'evento iniziale di B e la sua tupla abilitante sono riunite in un unico token nel posto `ActiveCopyInput`.

3. Lo scatto di `CreateActiveCopy` legge dal posto `Activities` (di tipo `Activity`) un token rappresentante la definizione di B, dal posto `Types` (di tipo `Metatype`) le definizioni dei tipi usati nella definizione di B e dl posto `ActiveCopyInput` l'evento iniziale e la sua tupla abilitante. Tale informazione è usata per creare un token rappresentante la nuova copia attiva dell'attività B. Il token così costruito è inserito nel posto `ActiveCopyToExecute` dallo scatto di `CreateActiveCopy`.

4. La black transition `SLANGInterpreter` ha come ingresso un token in `ActiveCopyToExecute`: l'esecuzione di tale transizione genera un process engine dedicato alla nuova copia attiva di B.

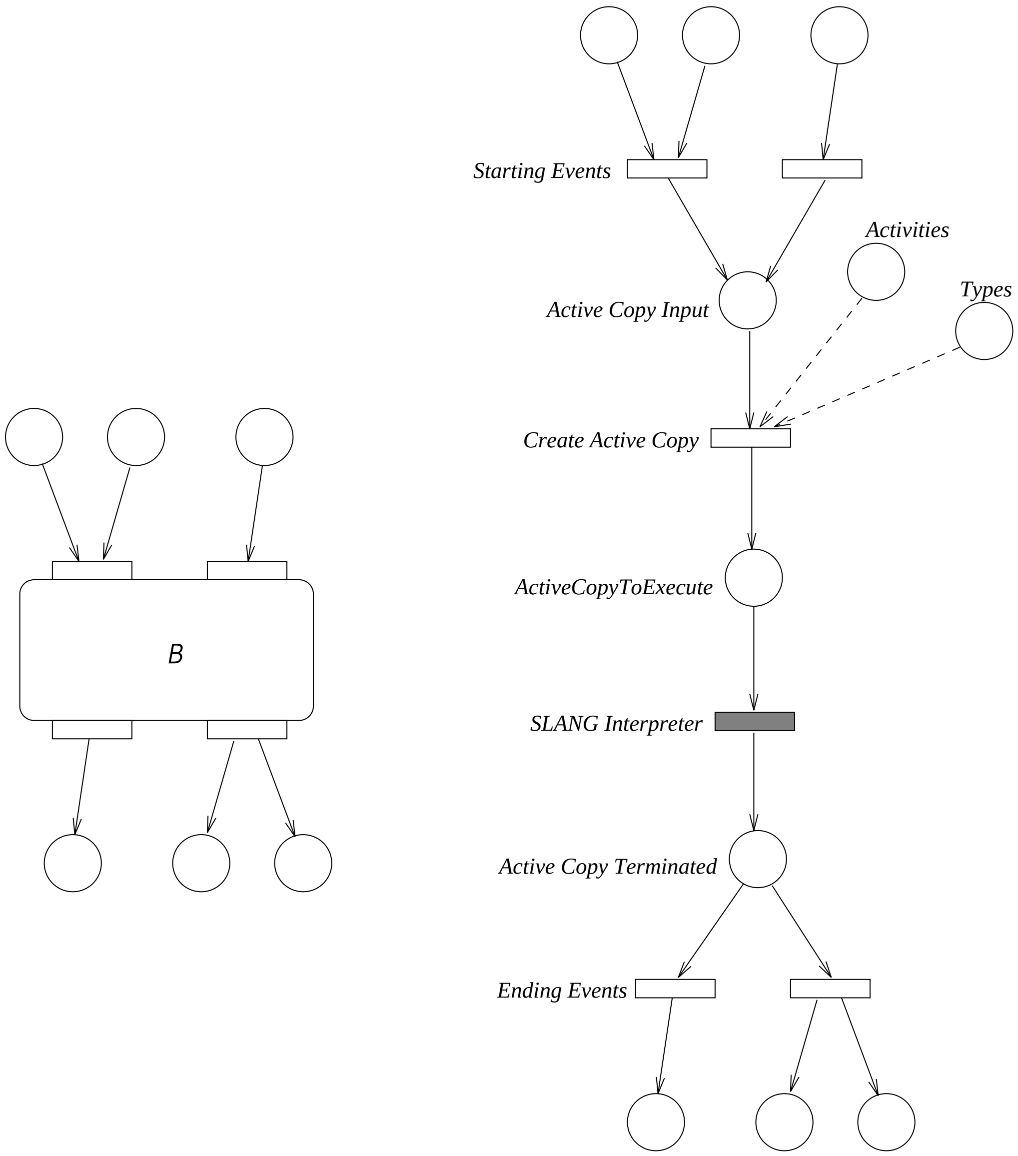

Figura 3.3: La rete corrispondente all'invocazione di un'attività.

5. Quando uno degli eventi finali della copia attiva di B si verifica, l'esecuzione termina e così la black transition che aveva generato il process engine, quindi la transizione genera un token in `ActiveCopyTerminated`. Tale token è di tipo `ActiveCopyType` e contiene informazioni sull'evento finale, insieme alla tupla di uscita ed allo stato finale della copia attiva.

6. Il valore della tupla di uscita è usato dagli eventi finali per determinare il contenuto dei posti di uscita dell'invocazione di attività corrente: la copia attiva è dunque distrutta e quindi il suo stato finale viene perso.

Dalla descrizione fornita si possono evincere alcune importanti considerazioni:

- L'invocazione di un'attività è collegata in maniera dinamica alla propria definizione (ed alla definizione dei tipi): questo meccanismo di *late binding* è la soluzione al problema dell'evoluzione del modello.

- Il meccanismo utilizzato per eseguire la definizione di un'attività attraverso l'interprete è lo stesso utilizzato per invocare un qualsiasi altro tool esterno.

- L'esecuzione delle attività è asincrona, cioè il chiamante continua la propria esecuzione mentre il chiamato è eseguito da un nuovo process engine. Inoltre, è possibile avere più esecuzioni simultanee della stessa definizione di attività, ciascuna delle quali gestita da un process engine dedicato.

- Il process engine che esegue l'attività può accedere a posti condivisi, che possono essere utilizzati per la sincronizzazione dell'esecuzione dei process engine.

Un importante vincolo imposto alle attività è che *un'attività può terminare la propria esecuzione soltanto se tutte le attività invocate hanno già terminato la loro.* Il soddisfacimento di questo requisito è garantito dal sistema.

È possibile definire a questo punto il concetto di *albero dinamico delle attività*, il quale risulta essere la controparte dinamica del DAG statico visto precedentemente. Dato un modello SLANG, $\mathcal{ACT}$ denota l'insieme di copie attive di tutte le attività in $ProcessActivities$ in un dato istante. Su questo albero può essere definita la relazione *è_invocato_da*, tale che B *è_invocato_da* A se e solo se B è stato creato da A. Tale relazione definisce una gerarchia dinamica che chiamiamo *albero dinamico delle attività*, la cui radice è la copia (unica) attiva dell'attività radice.

3.4.3 L'accesso allo stato di un'attività in esecuzione

Poiché i meccanismi descritti precedentemente sfruttano il late binding, è possibile istanziare un'attività con diverse definizioni di attività, manipolando la copia attiva: a tale scopo è però necessario sospendere l'esecuzione di tale copia attiva, pertanto sono necessari i meccanismi di sospensione e riattivazione delle attività che descriveremo in questa sezione.

Tali meccanismi sono ottenuti estendendo lo schema di Fig. 3.3 come mostrato in Fig. 3.4 e fornendo una trasformazione standard (in Fig. 3.5) da applicarsi alla definizione dell'attività che deve essere eseguita.

Ora possiamo riportare l'algoritmo utilizzato dall'interprete di SLANG, completo della possibilità di sospendere e riattivare le copie attive.

Nel creare una copia attiva, la transizione `CreateActiveCopy` in Fig. 3.4 inizializza il token in `ActiveCopyToExecute` come segue:

- Il campo `Act` della copia attiva viene riempito con la definizione di B estesa come specificato in Fig. 3.5.

- La transizione rimuove un token dal posto `DynTree` ed uno da `ActiveCopyInput`; tali token sono usati per creare un token da porre nel posto `Start` della copia attiva in corso di creazione. Tale copia possiede già uno stato, memorizzato nel campo `State` e costituito da tale token, il quale contiene anche le informazioni relative agli eventi e relative tuple di ingresso uscita.

Dunque, sia `a` il token rimosso dallo scatto della black transition `SLANG Interpreter` di Fig. 3.4: il process engine che viene creato esegue `a.Act` (cioè la rete di Fig. 3.5) con la marcatura iniziale contenuta in `a.State`. Nella marcatura iniziale, l'unica transizione abilitata è `StartActiveCopy`, poiché l'unico posto che contiene token è `Start`. L'azione di `StartActiveCopy` genera un identificatore univoco per la copia attiva e lo depone nel posto `ID`, inoltre genera un token nel posto condiviso `DynTree`, rappresentante il valore

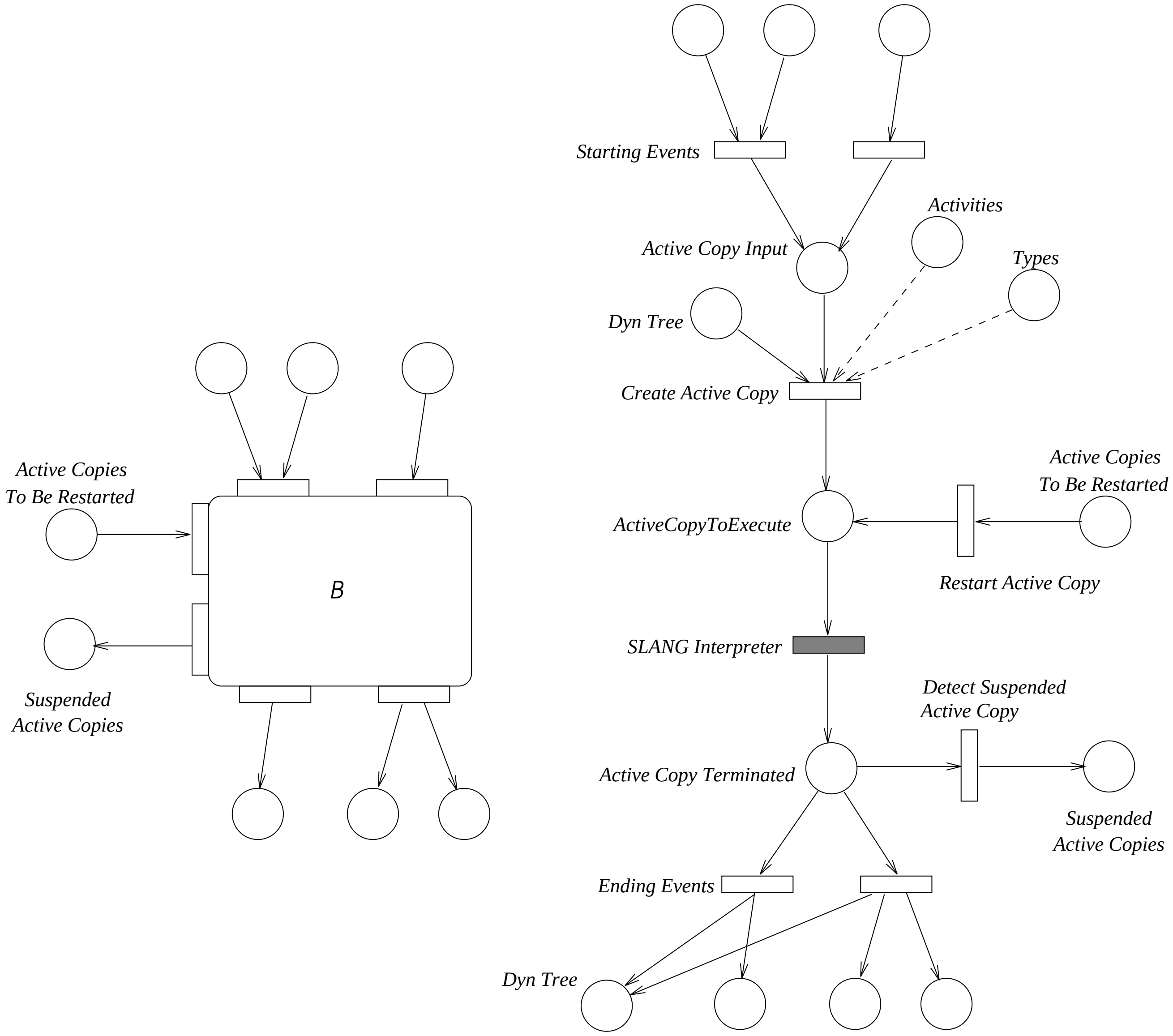

Figura 3.4: Estensioni allo schema precedente per tener conto della sospensione di un'attività.

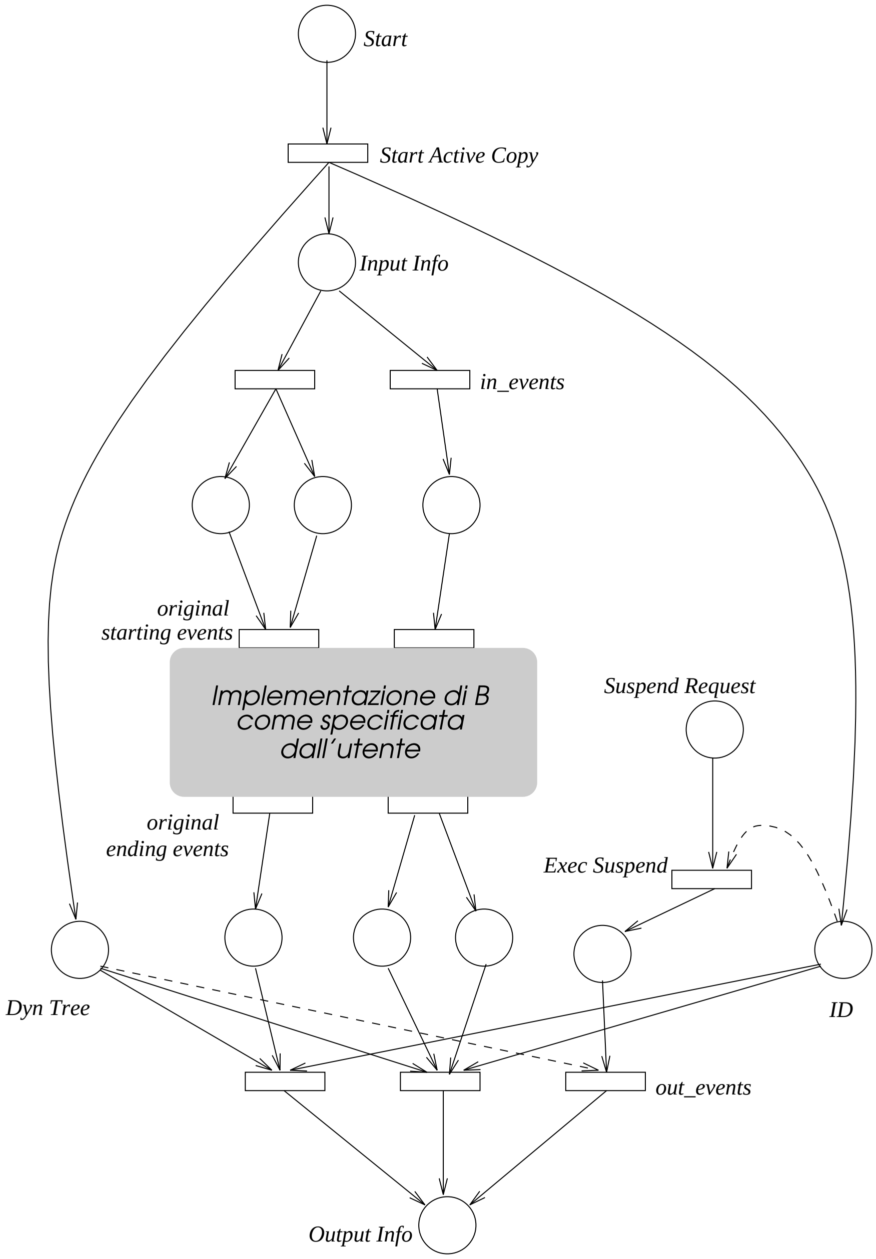

Figura 3.5: Definizione estesa di un'attività invocata, al fine di gestire gli aspetti di sospensione.

dell'albero dinamico delle attività dopo l'invocazione. Infine, genera un token contenente la tupla abilitante e l'evento d'inizio in `InputInfo`; tale token è usato da uno degli eventi in `in_events` per ricostruire la tupla abilitante per l'evento iniziale che deve scattare.

In un qualsiasi istante della propria evoluzione, una copia attiva può essere sospesa dallo scatto della transizione `ExecSuspend`. Tale transizione è abilitata quando un token contenente l'identificatore univoco della copia attiva da sospendere è inserito nel posto predefinito e visibile all'utente `SuspendRequest`. La copia attiva può terminare anche quando uno dei suoi eventi finali (in Fig. 3.5, "original ending events") scatta. In ambedue i casi, una delle transizioni in `out_events` scatta, producendo un token in `OutputInfo`. Se uno degli eventi appartenenti a `out_events` scatta come conseguenza dello scatto di uno degli eventi originali, i token contenuti in `DynTree` ed `ID` sono rimossi e il token posto in `OutputInfo` contiene informazioni circa l'albero dinamico, l'identificatore dell'evento originale che è scattato e le tuple di output. Se invece la terminazione avviene a causa di una sospensione, attraverso `ExecSuspend`, l'unica differenza è che l'albero dinamico questa volta non viene rimosso e non c'è tupla di output.

Lo scatto di uno qualsiasi degli eventi in `out_events` causa la terminazione del process engine. Come risultato di ciò viene creato un token nel posto `ActiveCopyTerminated` del chiamante; tale token (il cui tipo è `ActiveCopyType`) contiene la marcatura finale della copia attiva terminata. Il posto `OutputInfo` contiene informazioni sulla causa della terminazione: se tale informazione indica che la causa è stata una sospensione la transizione `DetectSuspendedActiveCopy` è abilitata allo scatto e la copia attiva viene mostrata all'esterno nel posto `SuspendedActiveCopies`.

Una volta che la copia attiva è stata depositata in questo posto, essa può essere manipolata come un qualsiasi altro dato del processo. Per ripristinare l'esecuzione di una copia attiva sospesa è necessario spostare la copia attiva (eventualmente modificata) da `SuspendedActiveCopies` a `ActiveCopiesToBeRestarted`. In questo modo, la transizione `RestartActiveCopy` sposta il token in `ActiveCopyToExecute` e quindi riprende il controllo "normale".

Una volta forniti i meccanismi base di sospensione e riattivazione delle istanze di attività, la gestione dell'evoluzione del modello spetta al modellatore del processo, che dovrà tra l'altro scegliere tra varie politiche possibili per la gestione del cambiamento. Non ci addentreremo in questi aspetti, sebbene importanti ai fini della gestione del processo; ciononostante per completezza riportiamo qui tre possibili politiche:

- *Politica "lazy":* non propaga i cambiamenti di tipi ed attività. Le nuove definizioni vengono utilizzate quando vengono create delle nuove istanze.
- *Politica "eager":* è basata su una propagazione immediata dei cambiamenti, anche alle istanze di attività già in esecuzione.
- *Politica "locale":* la modifica è relativa solo alla copia attiva in corso di modifica, nessuna definizione globale di tipo o attività viene cambiata, pertanto future istanze verranno create con la definizione originale.

Parte III

Il Manuale del Sistema di Qualità

Capitolo 4

Introduzione al Manuale del Sistema di Qualità

Il Manuale del Sistema di Qualità (o, più brevemente, Manuale di Qualità) utilizzato in Italtel BUTD Software [Ita88], ha svolto un duplice ruolo all'interno della nostra attività di specifica del processo software: da un lato esso ha rappresentato la fonte primaria di informazione per comprendere le linee essenziali del processo software utilizzato all'Italtel, dall'altro ha consentito di sviluppare un primo nucleo del modello interamente ed unicamente basato su di esso.

In ogni caso, la conoscenza degli aspetti di processo descritti nel Manuale di Qualità è strettamente necessaria per la comprensione della specifica SLANG descritta nei successivi capitoli.

Si fa presente che le informazioni sul Manuale di Qualità Italtel BUTD Software contenute in questo e in successivi capitoli sono utilizzabili nei previsti termini di legge.

4.1 Domini di applicazione ed obiettivi

Definizione. Il Manuale di Qualità Italtel BUTD Software stabilisce i *criteri gestionali*, gli *standard di progetto*, le *regole di produzione* e le *procedure operative* adottate per soddisfare i requisiti di un sistema di Controllo Qualità del Software al fine di assicurare la *qualità* del prodotto software e/o del servizio prestabiliti dal contratto.

Da questa definizione si può notare come l'obiettivo fondamentale attorno al quale è incentrato l'intero documento sia assicurare il servizio richiesto e la sua qualità e come ciò sia ottenuto attraverso diversi livelli di prescrizione, a partire da criteri di alto livello fino ad arrivare alle istruzioni operative, classificabili come di basso livello, in quanto descrivono in maniera assai dettagliata i vari passi necessari per svolgere un compito assegnato.

Il contenuto del Manuale di Qualità è il seguente:

- Il ciclo di vita del software.
- Le attività tecniche necessarie alla valutazione del progetto e del prodotto e alla definizione delle azioni correttive.
- Le responsabilità delle attività svolte all'interno delle Funzioni.
- L'identificazione del prodotto software con riferimento alle sue specifiche.
- L'elenco della documentazione prevista di produzione e della qualità, con indicazione dei responsabili della emissione.
- Le tecniche e gli strumenti utilizzati per la gestione della configurazione.
- I criteri e le metodologie di Verifica e Validazione e le fasi in cui essi sono utilizzati.
- I metodi e gli strumenti di valutazione, definizione ed esecuzione di azioni correttive.
- La gestione dei supporti.
- Le attività di qualificazione dei fornitori e di verifica delle forniture.
- I rapporti e gli audit della Funzione Qualità che documentano e valutano la qualità del processo di produzione e del prodotto software, e l'efficienza del sistema di controllo della qualità del software.

4.2 Struttura del Manuale di Qualità

Il Manuale di Qualità è organizzato in maniera gerarchica: immaginando di descriverne l'organizzazione mediante un albero, i capitoli posti alla radice ed ai primi livelli di tale albero contengono una descrizione maggiormente astratta del processo software, illustrandone i concetti generali e facendo ampio uso di definizioni, mentre i capitoli posti alle foglie dell'albero contengono le istruzioni operative e gli standard, i quali descrivono in maniera dettagliata i passi da compiere e le modalità della realizzazione dei compiti relativi.

È bene ricordare che il Manuale di Qualità è un oggetto fisicamente voluminoso, il che è del resto comprensibile vista la intrinseca complessità del processo che in esso è descritto.

Per questo motivo, come già spiegato altrove, si è scelto di modellare soltanto una parte, benché rilevante, del processo software Italtel BUTD evitando perciò di confrontarsi con l'intero Manuale di Qualità.

Per completezza, e per dare un'idea del tipo di documentazione fornitaci da Italtel, elenchiamo di seguito le parti di Manuale di Qualità in nostro possesso:

- *Qualità del software* (`A7060-W22-A805-*-35`)[1] Descrizione del Sistema di Qualità Software Italtel al livello di astrazione più alto.
- *Anomalie e modifiche del prodotto software* (`A7060-W22-A820-*-35`). Descrizione generale dei concetti base della gestione delle anomalie.
- *Audit del Sistema di Qualità Software* (`A7060-W22-A829-*-35`). Descrizione ad alto livello dell'attività di audit.
- *Istruzione operativa per la gestione della configurazione del prodotto software* (`A7060-W23-A880-*-35`). Descrizione dettagliata di come viene implementata la gestione della configurazione.

[1] Questo numero è il cosiddetto *part number*, usato per fornire un riferimento alla documentazione durante le operazioni di accesso all'archivio. Esso viene inoltre utilizzato anche all'interno del Manuale di Qualità quando è necessario riferirsi ad altre sezioni del documento.

- *Istruzione operativa per la gestione delle anomalie e modifiche del prodotto software* (`A7060-W23-A881-*-35`). Descrizione dettagliata di come viene implementata la gestione delle anomalie.

- *Istruzione operativa per l'effettuazione di attività di revisione del prodotto software* (`A7060-W23-A891-*-35`).
  Descrizione dettagliata dell'attività di revisione.

- *Standard del documento Rapporto di Lancio dei Test* (`A7060-W23-A844-*-35`). Descrive l'aspetto formale e sostanziale del documento.

- *Strumento PAPICS: manuale per la gestione della configurazione, delle anomalie e delle modifiche* (`A7060-W23-A883-*-35`).
  Contiene la descrizione degli automi a stati finiti che descrivono gli elementi di configurazione.

4.3 Descrizione e gestione dei ruoli

I ruoli ed i compiti all'interno della struttura organizzativa sono definiti come segue:

Funzione Produzione Software. È responsabile delle caratteristiche di progetto e sistema in termini di:

- Scelta dell'architettura, allocando le prestazioni a blocchi funzionali e definendo le interfacce tra tali blocchi, identificando le strutture di elaborazione e quindi la ripartizione tra le funzionalità hardware e software.

- Verifica della fase di integrazione e prove di sistema, da eseguirsi dopo la integrazione hardware/hardware e software/software. Per *integrazione* si intende l'attività di verifica di funzionalità del software caricato sull'hardware target nei vari sistemi di elaborazione che lo compongono.

Funzione Progettazione Software. È responsabile della progettazione e produzione del software, a partire dalle specifiche dei requisiti del prodotto software, in termini di articolazione delle seguenti fasi:

- specifica funzionale;
- definizione di architettura;
- realizzazione del prodotto software;
- definizione, realizzazione ed esecuzione delle prove di qualificazione.

A seconda della dimensione e complessità del progetto può essere composta da:

- **Sottofunzione Sviluppo Software**, a cui compete la realizzazione dei documenti e dei codici che costituiscono il prodotto software, e la partecipazione ai Team di Verifica e Validazione e alla Commissione di Valutazione (CCB).
- **Sottofunzione Qualificazione Software**, a cui competono la definizione, la preparazione e la esecuzione delle attività di qualificazione secondo le modalità riportate nelle istruzioni operative definite dalla Funzione Qualità e la partecipazione ai Team di Verifica e Validazione e alla Commissione di Valutazione (CCB).

Funzione Qualità. È responsabile della validazione e certificazione di conformità del prodotto software e della validazione di integrazione sul sistema. In particolare:

- Emette il Piano di Qualità del prodotto.
- Esegue attività di verifica e validazione per valutare la corretta operatività delle funzioni rispetto agli standard e alle procedure applicabili.
- Emette il Rapporto di Qualificazione, documento conclusivo delle attività di qualità.

Funzione Gestione Centro di Calcolo. È responsabile della gestione dei mezzi hardware di produzione e della gestione dei supporti fisici.

Funzione Supporti, nell'ambito della quale sono presenti:

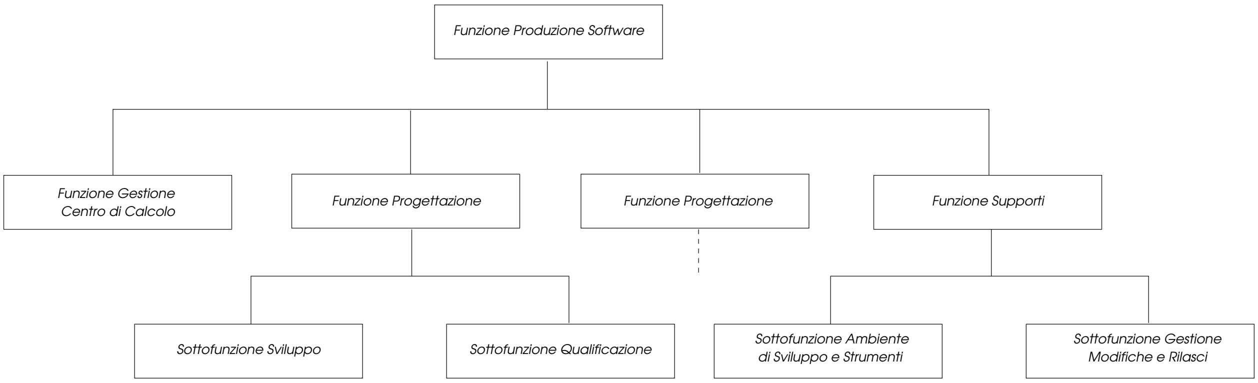

Figura 4.1: La struttura organizzativa di Italtel BUTD.

- **Sottofunzione Ambiente di Sviluppo e Strumenti**, a cui compete la gestione dei metodi e degli strumenti di sviluppo, la gestione degli strumenti di supporto alla qualificazione e la standardizzazione della scelta e dell'uso di metodi e strumenti.
- **Sottofunzione Gestione Modifiche e Rilasci**, a cui competono la gestione del prodotto, il controllo di configurazione ed il controllo delle modifiche.

La Fig. 4.1 mostra le relazioni tra i vari ruoli.

4.4 Documentazione software

Descriviamo qui la documentazione che deve essere prodotta durante l'esecuzione del processo software.

La comprensione di termini, definizioni e acronimi, sebbene possa risultare noiosa, è condizione necessaria per la comprensione della specifica SLANG.

4.4.1 Documenti di sviluppo soggetta a verifica e validazione

L'attività di verifica e validazione è *obbligatoria* sui seguenti documenti. I tempi ed i modi di tale attività verranno descritti nel seguito.

Requisiti di Sistema Software (SSR). Tale documento è il risultato dell'individuazione delle parti dell'intero prodotto che sono da sviluppare come prodotto software. Il documento indica il primo livello di architettura fisica del prodotto software, descrivendone i componenti, le loro corrispondenze con le aree funzionali e le relazioni tra essi. Descrive inoltre l'ambito operativo in cui il prodotto software dovrà essere impiegato e le funzioni di alto livello che questo dovrà svolgere verso l'ambiente circostante, incluse interfacce ed interdipendenze con il sistema hardware, i livelli prestazionali, i requisiti di affidabilità e manutenibilità e così via.

Requisiti Software (SR). Il documento contiene i riferimenti al documento di SSR e descrive i requisiti funzionali di una delle porzioni di sistema software identificate nel documento di SSR come componente software. In particolare tratta i seguenti temi:

- *aspetti utente*: singole funzionalità, modalità d'uso e di esecuzione, formati d'ingresso e uscita, messaggi d'errore, comandi, help e così via.
- *aspetti di progetto*: throughput, turnaround time, occupazione di risorse e simili.
- *aspetti di qualificazione*: relativamente a tali aspetti sono indicate le esigenze specifiche in termini di attributi di qualità e quelle relative alla fase di preparazione dei test funzionali.

Disegno di Architettura (AD). Questo documento contiene la descrizione della struttura fisica di alto livello di un componente software descritto nello specifico documento di SR: si noti in particolare che ad ogni AD corrisponde un solo SR. Il Disegno di Architettura riporta l'indicazione dei sottocomponenti individuati, le interfacce tra sottocomponenti, le descrizioni dettagliate di tutte le informazioni di ingresso ed uscita da ciascun sottocomponente, la descrizione dettagliata della trasformazione effettuata. In sintesi, è il documento descritto dalla specifica funzionale di ciascuno dei sottocomponenti individuati, al primo livello di decomposizione.

Disegno di Dettaglio (DD). Ha la stessa struttura del documento di Disegno di Architettura, di cui condivide principi e filosofia. Lo scopo di questo documento è quello di permettere la decomposizione di un prodotto in elementi più piccoli, dominabili dal punto di vista della dimensione e della complessità. Esso serve di riferimento per la fase di preparazione dei test di integrazione e di test di modulo.

Nel caso di prodotti di dimensioni limitate è ammesso l'accorpamento di documenti di AD e DD in unico documento: in questo caso, il documento nato dalla loro fusione viene indicato con la sigla AD-DD (Architectural and Detailed Design).

Modulo (Source Code Unit - SCU). Questo documento contiene il codice sorgente di una unità software individuata come la più piccola unità eseguibile (e testabile), scritta nel linguaggio di programmazione prescelto. Esso è costituito da **Unità di Compilazione (CU)**, ovvero da porzioni di codice compilabili separatamente. Ciascuna di queste Unità di Compilazione può essere ulteriormente articolata secondo le caratteristiche fornite dal linguaggio di programmazione.

Documentazione Utente (UD). È composta da uno o più manuali che descrivono:

- le modalità di installazione del prodotto nell'ambiente di esercizio;
- le modalità di impiego del prodotto software, orientato all'utente finale;
- le modallità di manutenzione del prodotto stesso.

CheckList (CKL). Questo tipo di documento ha lo scopo di permettere alla Sottofunzione Qualificazione Software di progettare correttamente i test e al Team di Verifica e Validazione di verificare che la quantità di test prevista è adeguata rispetto all'obiettivo. Le checklist sono differenziate per tipologia (funzionale, di integrazione, di modulo) e per livello (SR, AD, DD) e contengono in modo esplicito la descrizione di tutto ciò che i test devono esercitare, in termini di funzionalità, interfacce, dati, sequenze di esecuzione e così via.

Specifiche di Test (TS). Il documento descrive l'obiettivo del test, le sue modalità di funzionamento, con esplicito riferimento alle checklist, e la collocazione del test nella strategia generale di testing (precondizioni generate da altri e simili).

Codice di Test (TC). Il documento contiene, a seconda del caso, la definizione dei dati di test, il programma di test (codice sorgente) e/o le procedure di test (sequenze operative) non automatiche. Il Codice di Test è soggetto ad uno standard: non è soggetto ad una verifica e validazione diretta, ma è incluso nella revisione del Rapporto di Lancio di Test corrispondente.

Rapporto di Lancio di Test (TLR). Si tratta dei risultati dell'esecuzione di test, i quali raccolgono informazioni sul loro esito, descrivendo le condizioni di esecuzione, identificando i test eseguiti, quelli terminati con successo e quelli falliti, insieme ad eventuali anomalie riscontrate.

Il Piano di Sviluppo Software, Piano di Test, Piano di Configurazione appartengono alla classe di documenti soggetti a verifica e validazione, ma verranno descritti nella Sez. 4.5. La Fig. 4.2 mostra la gerarchia della documentazione descritta. Si noti che, a seconda della complessità e del'articolazione del progetto, possono essere presenti anche altri documenti di carattere prevalentemente operativo o gestionale, non soggetti a verifica e validazione, quali: Specifiche di Catena di Test, Piano di Dettaglio, Rapporto di Avanzamento.

4.4.2 Documenti di responsabilità della Funzione Qualità e soggetti a verifica e validazione

I documenti che ricadono in questa categoria sono essenzialmente due:

- **Piano di Qualità.** Verrà descritto nella Sez. 4.5.
- **Rapporto di Qualificazione.** Contiene le indicazioni degli esiti di tutte le operazioni di verifica e validazione effettuate sul prodotto durante lo sviluppo e la qualificazione le quali, facendo riferimento a tutti i documenti di verifica e validazione, permettono di valutare lo stato di qualità del prodotto e quindi la sua rilasciabilità.

Sono inoltre emessi dalla Funzione Qualità tutti i Rapporti di Verifica e Validazione (VVR) prodotti dal Team di Revisione, con la sola eccezione del

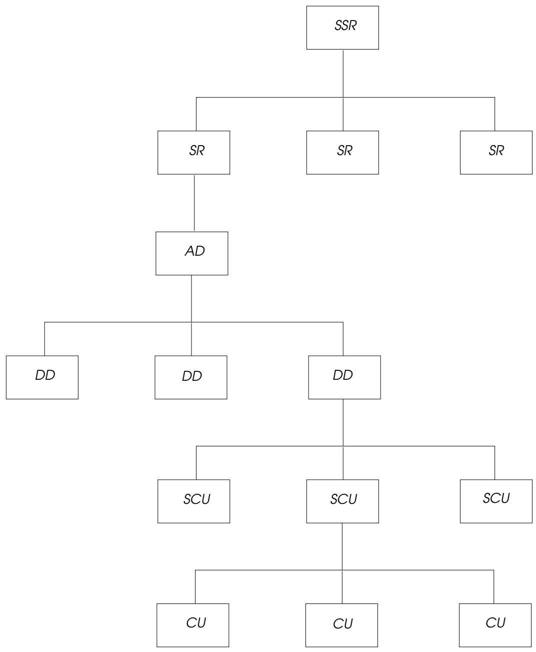

Figura 4.2: La struttura di un prodotto software Italtel.

rapporto relativo al Piano di Qualità, che viene invece emesso dalla Funzione Progettazione Software.

4.5 Pianificazione

Durante la fase iniziale del contratto viene redatto un **Piano di Qualità (QP)** contenente:

- Responsabilità delle varie attività allocate nella struttura organizzativa.
- Attività di verifica e validazione coerentemente con il Piano di Sviluppo del Software.
- Identificazione del prodotto software.
- Elenco dei documenti da emettere, indicandone le responsabilità.
- Gli standard, le tecniche e le convenzioni da adottare nello sviluppo e nella qualificazione.

- L'elenco delle attività di verifica e validazione da effettuare, allocate per elemento, descrivendo metodi e strumenti da adottare, ed eventuali vincoli da osservare per l'approvazione.
- L'indicazione delle linee guida della strategia di test.
- L'indicazione delle regole da adottare nella identificazione degli elementi del prodotto e nella gestione della configurazione.
- Le modalità di gestione delle anomalie e relative azioni correttive.
- Le procedure di gestione e conservazione dei supporti fisici.
- Il controllo del software di fornitura esterna.

Oltre alla stesura del Piano di Qualità, è prevista l'emissione dei seguenti documenti di pianificazione:

Piano di Sviluppo del Software (SDP). Esso identifica l'emissione dei documenti di progetto su scala temporale, individua le risorse umane, gli strumenti di sviluppo e le risorse di calcolo necessarie per un coerente sviluppo del programma. La responsabilità della stesura ed emissione è del responsabile della Funzione Progettazione Software.

Piano di Configurazione (CP). Tale documento identifica, all'interno dei criteri generali di gestione della configurazione, l'elenco di tutti gli elementi che compongono il prodotto software finale; in esso compare l'identificazione di tutti i moduli che dovranno essere sviluppati. Il documento nasce in versione parziale e cresce con la successiva articolazione e sviluppo del prodotto. La responsabilità della stesura ed emissione è del responsabile della Funzione Progettazione Software.

Piano di Test (TP). Il documento descrive la strategia di test. Esso definisce l'approccio, i metodi, gli strumenti adatti per ognuna delle attività di test e qualificazione previste. In particolare il documento descrive gli ambienti in cui i vari test sono eseguiti. La responsabilità della stesura ed emissione è della Sottofunzione di Qualificazione Software.

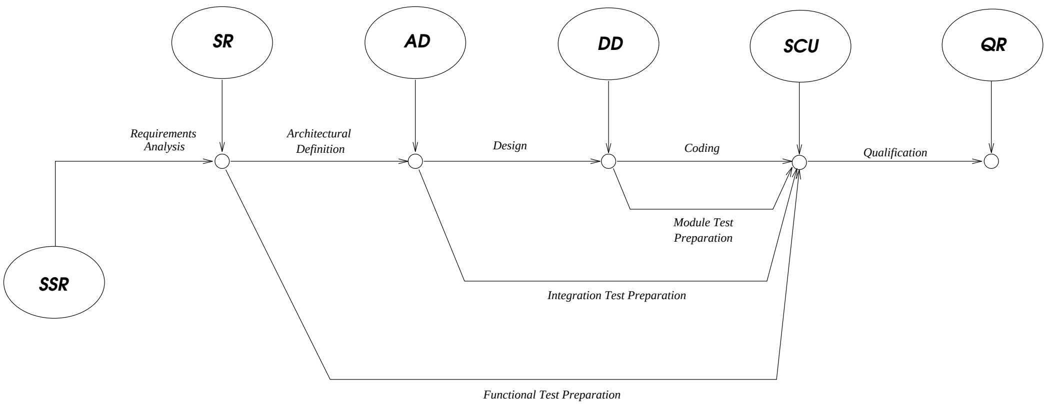

Figura 4.3: Descrizione del ciclo di vita adottato da Italtel BUTD.

4.6 Ciclo di vita del prodotto software

La Fig. 4.3 mostra il ciclo di vita adottato all'interno di Italtel BUTD Software. Come mostrato in tale rappresentazione, i flussi di attività si dividono in:

- sviluppo del software;
- preparazione dei test;
- qualificazione;
- verifica e validazione.

4.6.1 Attività di sviluppo

Tale attività viene suddivisa nelle seguenti quattro fasi:

- **Analisi dei requisiti (fase 1):** partendo dal documento Requisiti di Sistema Software si determinano uno o più documenti di Requisiti Software, i quali descrivono in dettagli i requisiti *funzionali* per ogni componente software.
- **Definizione dell'architettura (fase 2):** partendo dal documento di SR viene definita l'architettura di alto livello del componente software, indicando i suoi sottocomponenti e le interfacce tra essi.

- **Disegno dettagliato (fase 3):** partendo dal documento di AD vengono costruiti uno o più documenti DD, a loro volta scomponibili in sottocomponenti, fino a giungere all'individuazione di moduli.
- **Codifica (fase 4):** per ogni documento DD viene costruito il relativo insieme di SCU suddivisi fisicamente in unità di compilazione (CU).

4.6.2 Attività di preparazione dei test

Tale attività è eseguita attraverso una metodologia che permetta di determinare la copertura ottenuta mediante i test. La strategia di test è ripianificata per ogni progetto, al fine di definire la migliore strategia da adottare in funzione delle caratteristiche funzionali e strutturali del prodotto software in corso di sviluppo. I test sono preparati ed eseguiti da persone della Sottofunzion Qualificazione Software, diverse da chi ha prodotto la parte sotto test. In definitiva:

Preparazione dei test (fase 5): comprende la preparazione dei test funzionali (a partire dal SR), di integrazione (da AD e DD) e di modulo (a partire dai DD).

4.6.3 Attività di qualificazione

Tale attività prevede la seguente fase:

Qualificazione (fase 6): vengono eseguiti i test messi a punto nella fase precedente e viene costruito il Rapporto di Qualificazione. L'attività di qualificazione è costruita in maniera bottom-up a partire dall'esecuzione dei test di modulo, seguiti dai test di integrazione ed infine da quelli funzionali, producendo ad ogni passo il rispettivo TLR.

4.6.4 Attività di revisione

L'attività di revisione è eseguita in maniera concorrente rispetto al ciclo di sviluppo, praticamente dopo ciascuna delle fasi precedentemente illustrate. Nel seguito si farà riferimento ad alcuni termini chiave utilizzati all'interno di tale attività:

- **Revisione:** è l'attività formalizzata il cui compito è l'emissione di un documento di verifica e validazione relativo ad un elemento costituente un prodotto software.

- **Team di revisione:** è una commissione composta da rappresentanti delle Sottofunzioni Sviluppo e Qualificazione e della Funzione Qualità, che ha lo scopo di valutare gli elementi da revisionare e generare i corrispondenti Rapporti di Verifica e Validazione (VVR) ed eventuali Rapporti di Anomalia.

- **Verifica:** è l'attività di revisione che assicura che il risultato di una fase sia congruente con i requisiti definiti all'inizio della fase medesima.

- **Validazione:** è l'attività di revisione che assicura che il risultato di una fase sia congruente con i requisiti iniziali fissati.

4.7 Relazioni con altri standard

Italtel BUTD è un dipartimento che si occupa di software utilizzato per applicazioni di tipo militare, pertanto non deve meravigliare il fatto che il processo software adottato presenti molte analogie con quelli raccomandati in standard emessi dal Dipartimento della Difesa americano o dalla NATO, in particolare:

- *MIL-STD-2167a* [DOD88], emesso dal Dipartimento della Difesa americano.

- *AQAP-13* [AQA87], emesso dalla NATO.

- *ISO-9001-3* [ISO91], emesso dall'International Standard Organization.

Capitolo 5

La gestione delle anomalie

In questo capitolo viene illustrata la parte di Manuale di Qualità che si occupa della gestione delle anomalie. Si noti che quanto descritto si riferisce alla versione originale del manuale, senza tenere conto dei miglioramenti introdotti durante la nostra collaborazione con Italtel, spesso su nostra indicazione. Tali aspetti verranno esaminati nel Cap. 8.

5.1 Definizioni e concetti generali

Di seguito sono riportate alcune definizioni relative a concetti basilari per quanto riguarda l'attività di gestione delle anomalie software.

- **Anomalia.** È una irregolarità del prodotto software rispetto a quanto atteso. Generalmente emerge da attività di verifica e validazione o audit.

- **Commissione di Valutazione (Change Control Board).** È la commissione composta dai rappresentanti delle Sottofunzioni di Sviluppo, di Qualificazione e di Verifica e Validazione di Progetto. Ha lo scopo di esaminare i Rapporti di Anomalia (AR) emettendo Richieste di Modifica (RIMO).

- **Archivio di Sviluppo.** È l'archivio in cui sono registrati gli elementi costituenti il prodotto software in uso al progetto.

- **Archivio di Configurazione.** È l'archivio in cui sono registrati gli elementi costituenti il prodotto software presi in carico dal controllo configurazione.
- **Archivio dei Rilasci.** È l'archivio in cui sono registrati gli elementi costituenti il prodotto software rilasciato.

Le anomalie possono essere segnalate da un qualsiasi originatore, inclusi gli utenti, sia durante le fasi di produzione che di esercizio, ma frequentemente durante le fasi di verifica e validazione, in particolare:

- Come risultato negativo di una revisione.
  In tal caso il Team di Verifica e Validazione emette il VVR ed un Rapporto di Anomalia (AR) per ogni anomalia riscontrata.
- Come caso particolare dell'attività di verifica e validazione, durante il testing. In tal caso la Sottofunzione Qualificazione Software emette i TLR e un AR per ogni anomalia riscontrata.

Qualunque sia l'originatore, al fine di segnalare un'anomalia egli deve compilare un Rapporto di Anomalia e mandarlo alla Sottofunzione Gestione Modifiche e Rilasci (SGMR). Da questo punto in poi, l'anomalia segue dei passi ben precisi, dal momento che l'iter seguito per risolverla è definito mediante l'automa a stati finiti di Fig. 5.1. Si ricorda ancora che tale automa è stato modificato: la nuova versione verrà descritta nel Cap. 8.

Le transizioni di stato di ogni anomalia (e più in generale, di ogni elemento in controllo configurazione) vengono registrate all'interno del cosiddetto **Giornale degli Eventi**, il quale permette in sostanza di tenere traccia anche degli aspetti temporali legati agli elementi della configurazione e non solo delle varie versioni ed edizioni.

Verranno ora esaminati in dettaglio i passi necessari per la gestione di un'anomalia software, seguendo quanto descritto nella relativa istruzione operativa.

5.2 La gestione operativa dell'anomalia

La Fig. 5.2 mostra il flusso dei dati relativo alla gestione di un'anomalia software, mentre la Fig. 5.3 mostra il flusso di controllo. Tali figure vanno tenute presente durante la lettura successiva.

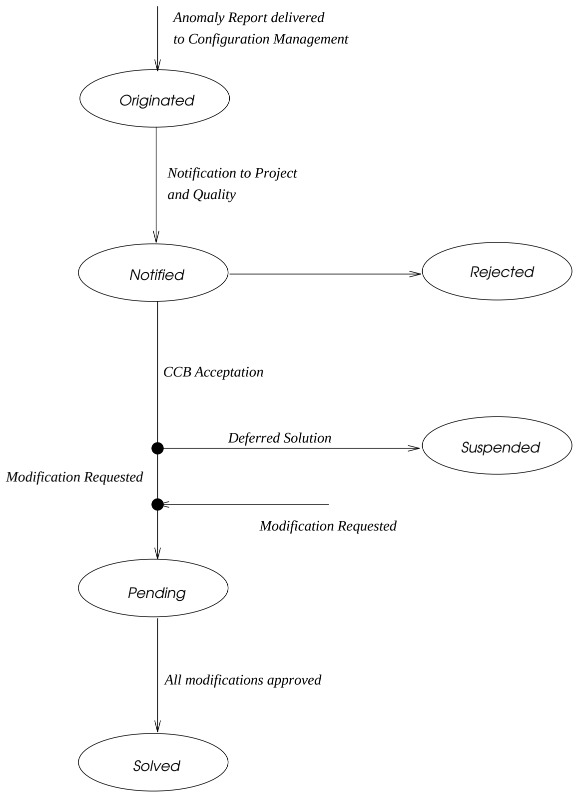

Figura 5.1: L'automa a stati finiti che descrive il Rapporto di Anomalia (versione originale).

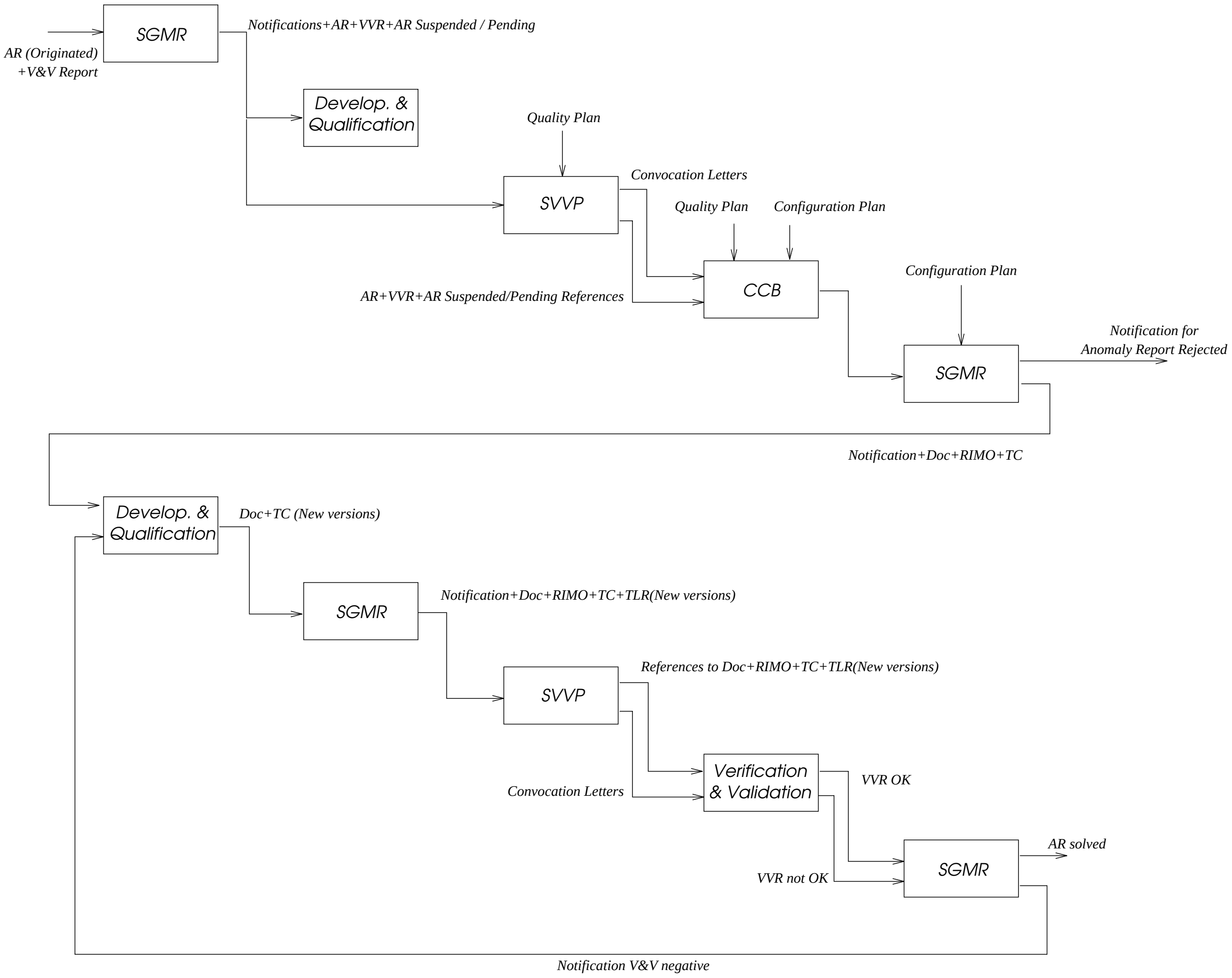

Figura 5.2: Il flusso di informazione nella gestione delle anomalie.

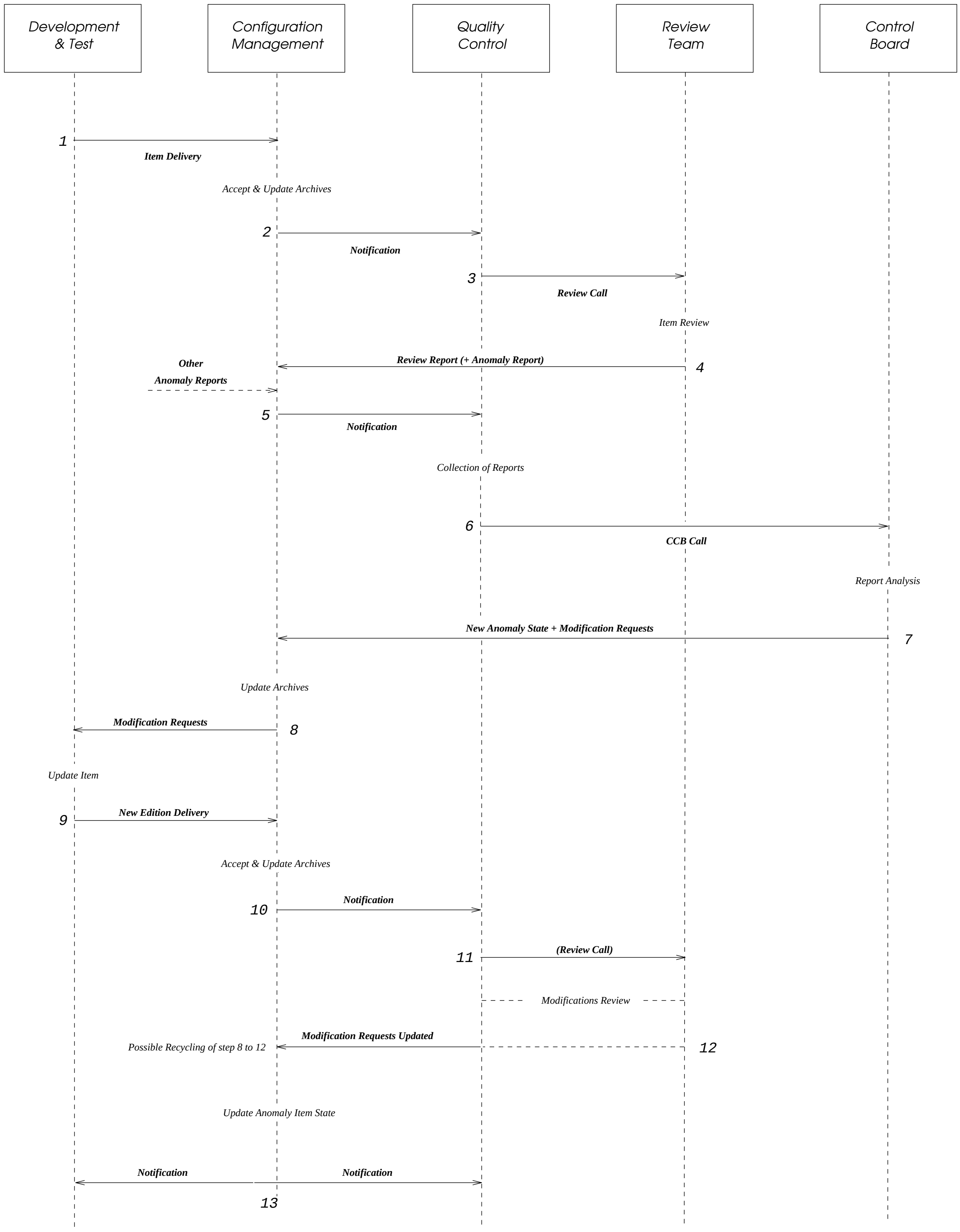

Figura 5.3: Il flusso di controllo nella gestione delle anomalie.

- La Sottofunzione Gestione Modifiche e Rilasci (SGMR), all'atto della consegna del Rapporto di Anomalia, effettua le seguenti operazioni:
  - Accetta o rifiuta la consegna effettuando controlli di coerenza col Piano di Configurazione e con le norme operative che regolano la consegna (correttezza del modulo).
  - Registra nell'archivio di Controllo Configurazione il Rapporto di Anomalia nello stato di 'Originated'.
  - Notifica al:
    * Sottofunzione di Sviluppo Software;
    * Sottofunzione di Qualificazione Software;
    * Sottofunzione di Verifica e Validazione di Progetto.

    La presenza di un'anomalia nello stato di 'Originated'. A seguito della notifica, il SGMR fa transitare l'anomalia nello stato di 'Notified'.
  - Fornisce il riferimento all'eventuale TLR o VVR associato al Rapporto di Anomalia notificato ed il riferimento a tutti gli AR ancora sospesi o pendenti alle Sottofunzioni indicate precedentemente, in modo da consentirne un'analisi integrata.
  - Aggiorna il Giornale degli Eventi.
- La Sottofunzione di Sviluppo e/o di Qualificazione, all'atto della notifica, hanno l'obbligo di avviare le necessarie azioni per la valutazione tecnica dell'anomalia. L'esito di tali azioni sarà discusso nell'ambito della Commissione di Valutazione (CCB).
- La Sottofunzione Verifica e Valutazione di Progetto procede alla convocazione del CCB. Per la formazione di esso, la SVVP fa riferimento al Piano di Qualità, sia del software che hardware. In particolare la composizione del CCB dipende dalla fase del ciclo di sviluppo in cui è stata originata l'anomalia e dalle caratteristiche del progetto. Tipicamente la Commissione può essere così composta:
  - analisti di sistema, sia software che hardware, soprattutto nelle fasi che comprendono la generazione dei documenti di SSR e SR;

  - il Committente, se previsto contrattualmente o comunque nei casi in cui l'anomalia riscontrata abbia effetti sui requisiti contrattuali;
  - i responsabili della Sottofunzione di Sviluppo e di Qualificazione interessati;
  - rappresentanti dell'ente che effettuerà l'integrazione hardware/software;
  - gli "utenti" degli elementi di configurazione coinvolti dalle anomalie.

  La convocazione della Commissione avviene per iscritto mediante un'apposita lettera di convocazione.

- La Commissione di Valutazione si riunisce per esaminare la/le anomalie. La riunione può terminare con uno dei seguenti esiti:

  - L'anomalia viene accettata, con formulazione di una richiesta di modifica. L'anomalia passa nello stato di 'Pending'. Le modifiche decise a soluzione dell'anomalia vengono descritte nei moduli di Richiesta Modifica (RIMO). Viene compilato un modulo per ogni elemento di configurazione da modificare. I moduli così compilati vengono consegnati al SGMR. Le modifiche richieste a fronte di un'anomalia 'Pending' richiedono una realizzazione immediata, o comunque a scadenza stabilita.
  - L'anomalia viene accettata ma la soluzione della stessa non viene stabilita; l'anomalia passa nello stato di 'Suspended'. La soluzione di un'anomalia 'Suspended' può essere rinviata ad una successiva versione/rilascio del prodotto software, oppure può essere rimandata ad una successiva riunione del CCB, qualora si individuino elementi necessari alla soluzione non immediatamente disponibili.
  - L'anomalia non viene riconosciuta come tale e pertanto risulta 'Rejected'.

  In ogni caso il CCB provvede, per ogni anomalia analizzata, all'aggiornamento del Rapporto di Anomalia, registrando l'esito della riunione. In particolare, viene aggiornato lo stato dell'anomalia e viene riportato nel rapporto il riferimento alle eventuali RIMO associate. Il Rapporto di Anomalia così aggiornato viene consegnato al SGMR.

- La Sottofunzione Gestione Modifiche e Rilasci effettua le seguenti operazioni nel momento in cui riceve il Rapporto di Anomalia:
  - Accetta o rifiuta la consegna effettuando controlli di coerenza col piano di Configurazione e con le norme operative che regolano la consegna (correttezza formale del modulo).
  - Aggiorna lo stato dell'anomalia nell'Archivio di Configurazione conformemente a quanto indicato sul Rapporto relativo come esito del CCB.
  - Se l'anomalia è stata accettata come 'Pending', registra nell'Archivio di Configurazione i moduli di RIMO associati.
  - Notifica la Sottofunzione di Sviluppo e la Sottofunzione di Qualificazione delle Richieste di Modifica ricevute.
  - Aggiorna il Giornale degli Eventi.
- Nel caso di anomalia accettata ed emissione dei moduli di RIMO, la Sottofunzione di Sviluppo e/o di Qualificazione, ricevendo la notifica della presenza di moduli di RIMO, devono:
  - Implementare le modifiche ai documenti e/o al codice descritte nei moduli di RIMO; per l'implementazione di tali modifiche viene effettuata una copia degli elementi di configurazione interessati dallArchivio di Configurazione all'Archivio di Sviluppo.
  - Consegnare al SGMR gli elementi di configurazione modificati.
  - Verificare l'efficacia delle modifiche apportate al software o codice di test. A tale scopo, nel caso di modifica al codice sorgente la Sottofunzione Qualificazione Software esegue i test definiti nell'ambito del CCB sul software già versato al SGMR. Il Rapporto di Lancio dei Test così generato viene consegnato alla Sottofunzione Gestione Modifiche e Rilasci. Nel caso di modifica al codice di test, i test modificati devono essere rieseguiti, a cura della Sottofunzione di Qualificazione, ed il relativo TLR deve essere consegnato al SGMR.
- La Sottofunzione Gestione Modifiche e Rilasci, all'atto della ricezione degli elementi modificati e dell'eventuale Rapporto di Lancio dei Test, effettua le seguenti operazioni:

  - Accetta o rifiuta la consegna effettuando controlli di coerenza col piano di Configurazione e con le norme operative che regolano la consegna (correttezza formale del modulo).
  - Aggiorna l'edizione degli elementi di configurazione che sono stati modificati e quella dell'eventuale Rapporto di Lancio dei Test ripetuti.
  - Notifica la Sottofunzione Verifica e Validazione di Progetto della presenza di elementi di configurazione che necessitano di un'attività di revisione. Contestualmente alla notifica invia una copia del Rapporto di Lancio di Test, se disponibile.
  - Aggiorna il Giornale degli Eventi.

- La Sottofunzione Verifica e Validazione di Progetto provvede alla convocazione del Team di Verifica e Validazione. La composizione di tale team si riferisce a quella reltaiva alla revisione delle edizione precedente dei documenti in esame, con l'inclusione degli autori delle RIMO relative.

- Il Team di Verifica e Validazione effettua l'attività di revisione analizzando la documentazione disponibile e in particolare il Modulo di Richiesta Modifica, la versione aggiornata dell'elemento di configurazione in esame e gli eventuali Rapporti di Lancio dei Test. Sulla base dell'esame effettuato il Team di Verifica e Validazione decide se la RIMO è stata soddisfatta. In particolare una RIMO può dirsi risolta quando sono state effettuate tuttte le modifiche all'elemento di configurazione previste e si è dimostrata l'efficacia delle modifiche apportate.

  Come effetto della riunione del Team di Verifica e Validazione viene emesso e consegnato al SGMR un Rapporto di Verifica e Validazione sull'elemento di configurazione esaminato.

- Il SGMR, all'atto della ricezione del VVR effettua le seguenti operazioni:

  - Accetta o rifiuta la consegna effettuando controlli di coerenza col piano di Configurazione e con le norme operative che regolano la consegna (correttezza formale del modulo).

- Archivia il Rapporto di Verifica e Validazione sotto controllo configurazione nello stato 'Approved'.
- Se l'attività di verifica e validazione si è conclusa con esito positivo provvede all'aggiornamento dello stato o della edizione dell'elemento modificato. Quando l'esito è positivo si possono verificare rispettivamente i seguenti casi:
  * Caso di anomalia emersa a fronte di revisione su un elemento di configurazione in stato di 'Draft': l'elemento passa nello stato di 'Approvato'.
  * Caso di anomalia emersa su un elemento 'Approved' e/o in fase di test o di uso: viene generata una nuova edizione dell'elemento di configurazione.
- Notifica le funzioni interessate della realizzazione della modifica.
- Nel caso in cui le RIMO siano state risolte, analizza lo stato delle Anomalie associate: se tutte le RIMO associate ad una anomalia sono risolte anche l'anomalia risulta 'Solved'.
- Se l'attività di verifica e validazione si è conclusa con esito negativo aggiorna l'edizione dell'elemento modificato e provvede ad avvisare la Sottofunzione di Sviluppo e/o la Sottofunzione di Qualificazione autori della modifica dell'esito negativo dell'attività di verifica e validazione effettuata. L'anomalia e le relative RIMO restano nello stato precedente.
- Aggiorna il Giornale degli Eventi.

Capitolo 6

La gestione della configurazione

In questo capitolo vengono presentati gli aspetti del Manuale di Qualità relativi alla gestione del controllo di configurazione. Anche in questo caso, come del resto per la gestione delle anomalie, si riportano solo gli aspetti presenti nella versione originale del Manuale di Qualità: ulteriori aspetti emersi durante la realizzazione della specifica e inclusi in essa verranno presentati nel Cap. 8.

6.1 Definizioni e concetti generali

Definizione. Scopo della Gestione della Configurazione è l'identificazione e la gestione degli elementi che costituiscono il prodotto software.

In particolare per *identificazione* s'intende l'attribuzione di un identificatore univoco ai documenti e al codice che costituiscono il prodotto software, per *gestione* l'applicazione ai documenti ed al codice di istruzioni operative che consentono in ogni istante di conoscere l'insieme degli elementi "ufficiali" che costituiscono il prodotto software e il loro stato, così da garantire che le funzioni coinvolte nello sviluppo del prodotto operino sempre su elementi aggiornati e consistenti.

A tal fine è necessario definire:

- le modalità di consegna, alla Sottofunzione Gestione Modifiche e Rilasci, degli elementi che costituiscono il prodotto software;
- le modalità di attribuzione degli identificatori ai documenti ed al codice;
- il flusso di informazioni relativo alle attività di revisione sugli elementi di configurazione, nell'ambito della realizzazione del prodotto software;
- il flusso di informazioni relativo all'attività di qualificazione del prodotto software;
- il flusso di informazioni relativo all'attività di gestione delle anomalie e delle modifiche;
- le modalità con cui effettuare il rilascio finale del prodotto software;
- la struttura e le modalità di aggiornamento del Giornale degli Eventi sul quale devono essere registrati tutti gli eventi (consegne, modifiche, rapporti di anomalia e così via) che hanno luogo durante lo sviluppo del prodotto software e che interessano la gestione delle configurazioni.

6.2 La gestione operativa del controllo configurazione

In questa sezione vengono riportati i passi che devono essere eseguiti durante l'esecuzione del controllo configurazione. In particolare vengono distinti i due aspetti di gestione dei documenti e del codice e della gestione dei test. Per l'illustrazione di tali aspetti vengono utilizzati diagrammi simili a quelli già utilizzati per la descrizione operativa della gestione delle anomalie.

6.2.1 La gestione dei documenti e del codice

La Fig. 6.1 mostra il flusso di informazioni necessario a gestire i documenti ed il codice in controllo configurazione. Di seguito sono riportati i dettagli operativi della gestione:

- A partire dai documenti di riferimento per la fase specifica, la Sottofunzione di Sviluppo o la Sottofunzione di Qualificazione realizzano il

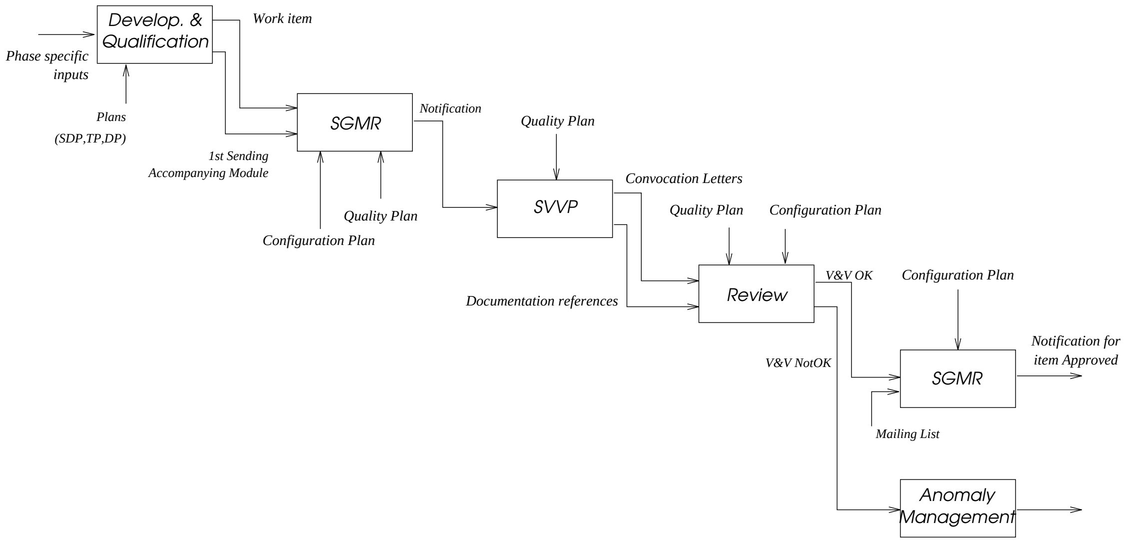

Figura 6.1: Il flusso di informazioni relativo alla gestione dei documenti e del codice.

documento previsto. A completamento della fase, tale documento viene consegnato alla Sottofunzione Gestione Modifiche e Rilasci, insieme al Modulo di Prima Consegna.

- All'atto della consegna, il SGMR effettua le seguenti attivitá:
  - Accetta o rifiuta la consegna effettuando controlli di coerenza con il Piano di Configurazione e con le norme operative che regolano la consegna (correttezza del modulo di accompagnamento).
  - Assegna al documento lo stato di 'Draft'.
  - Registra il documento ricevuto nell'archivio di configurazione.
  - Notifica la Sottofunzione di Verifica e Validazione di Progetto dell'avvenuta consegna e conseguente richiesta di revisione. Contestualmente invia copia del documento oggetto di verifica e validazione.
  - Aggiorna il Giornale degli Eventi.
- All'atto della notifica, la SVVP effettua le seguenti attività:

  - Procede alla convocazione del Team di Revisione, composto in maniera analoga a quanto visto per la gestione delle anomalie.
  - Assicura, come previsto dal Piano di Qualità, che sia disponibile la documentazione di supporto per la revisione.

- Il Team di Review effettua la revisione. Come risultato di tale revisione, il rappresentante della Funzione Qualità emette il Rapporto di Verifica e Validazione relativo al documento in esame e lo consegna alla SGMR insieme agli eventuali Rapporti di Anomalia emersi dalla revisione.

- Il SGMR, all'atto della consegna, effettua le seguenti attività:
  - Accetta o rifiuta la consegna effettuando controlli di coerenza con il Piano di Configurazione e con le norme operative che regolano la consegna (correttezza del modulo).
  - Archivia il Rapporto di Verifica e Validazione con lo stato di 'Approved' e gli eventuali AR nello stato di 'Originated'.
  - Nel caso in cui il Rapporto di Verifica e Validazione indichi esito positivo aggiorna lo stato del documento in 'Approved' e notifica tutti gli enti interessati dell'avvenuta revisione con esito positivo, utilizzando l'apposita *Mailing List* [1] gestita dal SGMR.
  - Nel caso in cui il Rapporto di Verifica e Validazione indichi esito negativo innesca la procedura di gestione delle anomalie.
  - Aggiorna il Giornale degli Eventi.

6.2.2 La gestione dei test

La Fig. 6.1 mostra il flusso di informazioni necessario a gestire i documenti ed il codice in controllo configurazione. Di seguito sono riportati i dettagli operativi della gestione:

- La Sottofunzione Qualificazione, sulla base di quanto pianificato nel Piano di Test esegue i test contenuti nei documenti Codici di Test(TC) sui Moduli(SCU) che si trovano nello stato 'Approved' ed inoltre sui sottocomponenti e componenti derivati dall'integrazione dei Moduli.

[1] Sono le liste di persone a cui spedire la notifica ogniqualvolta avviene una transizione di stato sull'elemento.

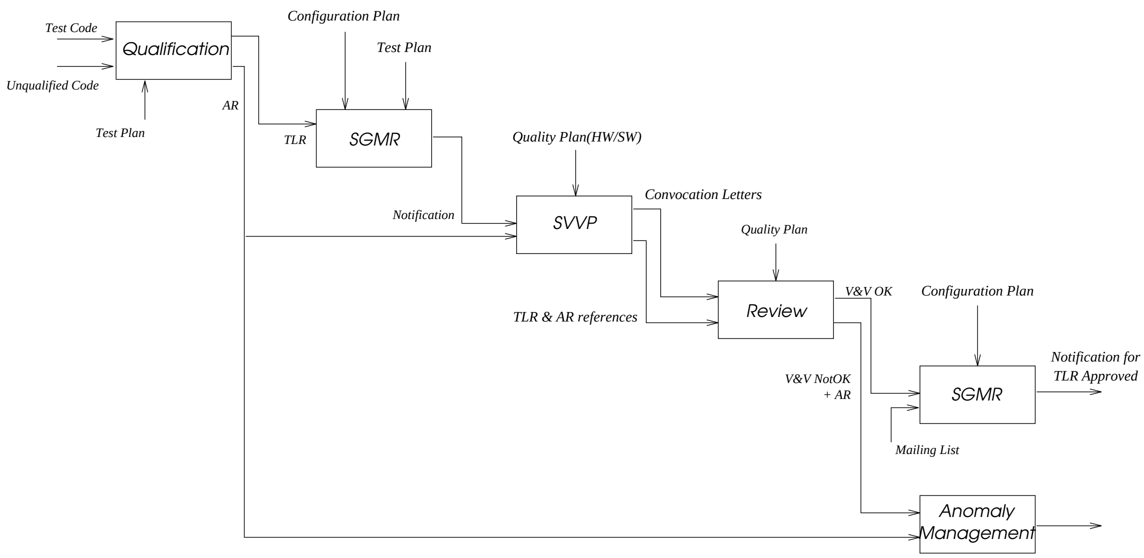

Figura 6.2: Il flusso di informazioni relativo alla gestione dei test.

- La sottofunzione di Qualificazione conclude l'esecuzione dei test emettendo il Rapporto di Lancio dei Test (TLR). Nel caso in cui uno o più test abbiano avuto esito negativo, la Sottofunzione emette il Rapporto di Anomalia relativo. TLR ed eventuali AR sono consegnati al SGMR il quale provvede, in presenza di AR, a innescare la gestione delle anomalie.

- La SGMR, all'atto della ricezione dei TLR e degli eventuali AR effettua le seguenti attività:

  - Accetta o rifiuta la consegna effettuando controlli di coerenza con il Piano di Configurazione e con le norme operative che regolano la consegna (correttezza dei moduli).
  - Registra nell'archivio di configurazione gli eventuali AR, nello stato di 'Originated'.
  - Notifica la Sottofunzione di Verifica e Validazione di Progetto della presenza di TLR che necessitano di revisione.
  - Aggiorna il Giornale degli Eventi.

- La SVVP all'atto della notifica effettua le seguenti operazioni:

  - Provvede alla convocazione del Team di Review, con composizione definita in base al Piano di Qualità, analogamente a quanto già visto precedentemente.
  - Assicura, come previsto dal Piano di Qualità, che sia disponibile la documentazione di supporto alla revisione.

- Il Team di Verifica e Validazione effettua la revisione del TLR secondo le modalità descritte nel Piano di Qualità. Tale attività può concludersi con:

  - *Esito positivo.* Viene emesso e consegnato alla SGMR il documento Rapporto Verifica e Validazione del Rapporto di Lancio dei Test (TLVVR).
  - *Esito negativo.* Vengono emessi e consegnati al SGMR il TLVVR ed il relativo AR.

- La Sottofunzione Gestione Modifiche e Rilasci all'atto della consegna effettua le seguenti operazioni:

  - Accetta o rifiuta la consegna effettuando controlli di coerenza con il Piano di Configurazione e con le norme operative che regolano la consegna (correttezza dei moduli).
  - Registra nell'archivio di configurazione il TLVVR e gli eventuali AR negli stati, rispettivamente, 'Approved' e 'Originated'.
  - Aggiorna opportunamente gli stati del TLR e del documento testato.
  - Aggiorna il Giornale degli Eventi.

Capitolo 7

Problemi sollevati dal Manuale di Qualità

In questo capitolo vengono descritti i problemi che si sono dovuti fronteggiare durante il lavoro di modellazione del processo software Italtel BUTD.

7.1 Tipologie di problemi

Il discorso viene focalizzato sul Manuale di Qualità in quanto quest'ultimo ha svolto, ai fini del nostro lavoro, una duplice funzione: da un lato esso ha fornito il riferimento principale per la creazione di un contesto di informazioni in cui muoversi, in termini di comprensione del processo software, sia in termini di linee generali che di dettagli operativi, dall'altro ha costituito l'oggetto principe della modellazione, permettendo una prima realizzazione totalmente incentrata su tale documento.

Sebbene quest'ultimo risultato sia stato possibile grazie alle doti di chiarezza e completezza evidenziate dal Manuale di Qualità, è nostra opinione che sia possibile identificare le seguenti caratteristiche del documento che necessitano di ulteriori sforzi da parte degli estensori:

1. **Ambiguità.** Il Manuale di Qualità è scritto per la maggior parte in linguaggio naturale, pertanto è intrinsecamente ambiguo. Un livello di formalizzazione è introdotto mediante l'utilizzo di diagrammi SADT per specificare i flussi di informazioni e macchine a stati finiti per specificare il controllo su elementi di configurazione e anomalie.

2. **Inconsistenza.** Il documento che costituisce il Manuale di Qualità è un oggetto fisicamente assai voluminoso, pertanto non dovrebbe stupire l'affermazione che la gestione di tale documento risulta assai complessa e altamente soggetta ad inconsistenze, soprattutto quando sia necessario modificare parti di Manuale, specie se scritte da persone diverse in tempi diversi. È stato constatato che tali inconsistenze sono ancora presenti, nonostante il Manuale di Qualità venga revisionato su base annuale da parte della Funzione Qualità.

3. **Incompletezza.** Alcuni aspetti del processo software, spesso di rilevante importanza, vengono lasciati indefiniti o descritti ad un livello di dettaglio non soddisfacente a fornire una vera comprensione del problema in esame.

4. **Migliorabilità.** Alcune parti della descrizione possono essere riformulate per meglio organizzare la descrizione oppure per ovviare ai problemi descritti nei punti precedenti. In aggiunta a questo tipo di modifiche, si deve tenere conto del fatto che anche il processo descritto all'interno del Manuale di Qualità è passibile di miglioramenti formali e sostanziali i quali, una volta vagliati ed approvati, devono essere inseriti al più presto all'interno della descrizione ufficiale. Esempi di miglioramenti scaturiti dal lavoro di modellazione e inglobati in nuove versioni del Manuale di Qualità sono descritti nel Cap. 8.

Va tenuto presente che l'enfasi dei problemi discussi è sulla descrizione e non sul processo: ciò significa che ad esempio una descrizione incompleta può generare una definizione ambigua del processo.

L'uso di uno strumento formale di descrizione del processo software, quale è SLANG, concorre alla soluzione dei problemi elencati precedentemente, infatti:

1. Una notazione formale, dove con *formale* si intende definita mediante regole di tipo logico o matematico, in maniera univoca, evita problemi di ambiguità. In particolare, la sintassi e la semantica di SLANG seguono una sintassi ed una semantica definite in maniera rigorosa ed univoca [Lip93].

2. Una notazione formale di per sè non evita problemi di inconsistenza: permette però la realizzazione di sistemi di controllo automatico della consistenza delle informazioni che costituiscono la specifica del processo software. SLANG, ad esempio, sfrutta i meccanismi di type checking e le regole di visibilità relative alle attività, le quali permettono un controllo automatico, mentre tools di analisi più sofisticati sono allo studio.

3. In generale, una notazione formale impone al suo utilizzatore una metodologia da seguire durante lo sviluppo di una specifica che ne faccia uso: questo tipo di metodologia, insieme con i costrutti linguistici della notazione, possono aiutare l'estensore della specifica ad evitare problemi di incompletezza. Per esempio, un'attività SLANG costringe a specificare chiaramente gli input e gli output dell'attività medesima: questo non significa ad esempio che la rete che descrive il comportamento dell'attività sia specificata in maniera completa, ma certamente evita almeno il caso (peraltro abbastanza comune) in cui di un'attività vengono specificati i dettagli operativi ma viene tralasciata la descrizione dei prodotti finali.

4. La possibilità di migliorare il processo può essere aumentata in maniera drastica dall'utilizzo di notazioni formali di tipo grafico, quali lo stesso linguaggio SLANG. È stato sperimentato infatti che risulta in generale molto più immediata la comprensione di un diagramma rispetto alla comprensione di una serie (spesso lunga) di descrizioni scritte in linguaggio naturale.

I problemi descritti precedentemente sono stati discussi insieme all'esperto del processo software, Ing. Maurizio Loi: quando possibile quest'ultimo ci ha fornito la corretta interpretazione o le informazioni mancanti nel Manuale di Qualità; in altri casi ha preso atto del problema in quanto tale, riservandosi di proporre nell'immediato futuro una soluzione all'interno di nuove versioni del Manuale.

7.2 Aspetti di ambiguità

Durante il nostro lavoro, sono stati incontrati spesso problemi che sembravano dovuti ad un qualche tipo di ambiguità all'interno della descrizione del

Manuale di Qualità. Non tutti questi problemi sono stati poi identificati come problemi effettivi del Manuale, per varie ragioni:

- La parte di Manuale in nostro possesso è, come abbiamo già accennato, limitata, pertanto alcune volte ciò che poteva essere considerato un'ambiguità, era in realtà chiaro in un contesto definito altrove.

- Il Manuale di Qualità è rivolto ad un lettore che abbia già una certa idea del contesto operativo industriale in cui è inserito, condizione non verificata nel nostro caso, pertanto alcune ambiguità non erano in realtà più tali a patto di possedere effettivamente il necessario insieme di nozioni "di base".

- Alcune parti di processo non sono descrivibili nei minimi particolari, o almeno ciò non rientra negli obiettivi attuali del Manuale di Qualità: alcune di queste parti sono state scambiate per ambiguità all'interno della descrizione.

Un esempio di reale ambiguità è costituito dal fatto che all'interno della gestione delle anomalie software non è chiaramente evidenziato un differente comportamento tra i passi da compiere per un documento di design anomalo ed un modulo software, laddove risulta invece evidente che in generale le operazioni da compiere sono sicuramente diverse. Si noti che la specifica SLANG ha rimosso questo tipo di ambiguità, definendo le azioni da intraprendere nei differenti casi.

7.3 Aspetti di inconsistenza

Le inconsistenze tra diverse parti di Manuale hanno costituito uno dei maggiori ostacoli alla comprensione del documento: le cause di ciò, come già evidenziato, risiedono nei problemi presenti nel'aggiornamento coerente di un documento di tali proporzioni.

Il problema si manifesta con riferimenti a concetti o dettagli operativi con differenze più o meno marcate a seconda della sorgente dalle quali sono prelevate: a titolo di esempio, spesso i diagrammi SADT riportati anche nei Cap. 5 e 6 descrivono input ed output differenti o mancanti rispetto a quelli descritti nella descrizione testuale fornita all'interno del Manuale.

7.4 Aspetti di incompletezza

Per gli aspetti di incompletezza vale quanto già affermato nella Sez. 7.2 a proposito delle ambiguità: spesso ciò che in primo momento si era ritenuto incompleto non lo era per uno dei motivi già descritti precedentemente.

Vale la pena di fornire qualche esempio di aspetti descritti in maniera incompleta all'interno del Manuale di Qualità:

- Quando un azione intrapresa da un attore del processo viene giudicata con esito negativo da altri attori (ad esempio, un modulo scorretto viene rifiutato dal SGMR, od un elemento di qualificazione che non passa la riqualificazione), non viene detto esplicitamente che è necessaria una notifica all'originatore.

- Spesso alcune informazioni sono "nascoste": per esempio, non si parla praticamente mai in maniera esaustiva degli aspetti temporali, i quali rivestono invece un ruolo chiave nella gestione del progetto, anche e soprattutto dal punto di vista economico.

- Non sono indicati i metodi (o i riferimenti ad essi) con cui viene provata l'efficacia delle modifiche apportate al codice.

- Nella versione originale del Manuale non era stata fornita nessuna informazione su come deve evolvere una RIMO. Ciò è stato incluso in successive versioni anche grazie all'apporto del nostro lavoro.

Capitolo 8

Aspetti del processo non inclusi nel Manuale di Qualità

Nel Cap. 7 è stato evidenziato come attualmente esistano sicuramente ampi margini di miglioramento all'interno del Manuale di Qualità, non solo per quanto riguarda gli aspetti di descrizione del processo, ma anche e soprattutto per quanto riguarda il processo medesimo.

La letteratura scientifica ([ABGM92], [Kel89], [Hum89]) ha evidenziato che uno dei benefici generalmente attesi da una qualsiasi attività di formalizzazione del processo di produzione del software è la possibilità di individuare, oltre che problemi analoghi a quelli descritti nel Cap. 7, anche ottimizzazioni e raffinamenti del processo medesimo.

La nostra attività non ha fatto eccezioni in questo senso, in quanto un certo numero di nostri suggerimenti sono stati fatti propri da Italtel e verranno probabilmente inclusi in nuove versioni del Manuale di Qualità, in particolare le modifiche agli automi che descrivono il controllo applicato ai Rapporti di Anomalia, alle Richieste di Modifica ed in generale alle varie tipologie di elementi di configurazione.

Accanto a questi aspetti, nel capitolo sono presentati anche alcune informazioni non contenute nel Manuale di Qualità, ma evidenziate attraverso colloqui con l'esperto di processo ed incluse nella modellazione.

8.1 I nuovi automi a stati finiti

Come già descritto precedentemente, nel Manuale di Qualità sono presenti un certo numero di automi a stati finiti che descrivono gli stati che un elemento in controllo configurazione può attraversare durante il suo sviluppo. Nella versione del Manuale a noi fornita, gli automi presenti erano sostanzialmente due, e cioé l'automa che descrive il controllo per un Rapporto di Anomalia ed un automa generico per ogni tipo di elemento di configurazione appartenente al prodotto software, sia esso un documento oppure un codice sorgente.

In realtà, all'interno del processo effettivamente utilizzato da Italtel BUTD "sul campo", all'atto della nostra modellazione erano già utilizzati altri tipi di automi: un automa per la descrizione delle RIMO era già impiegato, benché assai semplificato, mentre si era già cercato di distinguere gli oggetti appartenenti al prodotto software in varie classi:

- Codice, ovvero moduli software, ovvero SCU.
- Rapporti di Lancio dei Test.
- Documenti non soggetti a verifica e validazione.
- Documenti soggetti a verifica e validazione.

a ciascuna delle quali veniva fatto corrispondere un diverso automa.

Il nostro apporto si è concretizzato nella modifica e nell'estensione dei due automi di AR e RIMO, in modo da renderli effettivamente corrispondenti alla realtà delle sequenze operative utilizzate nel processo, ed uniformare i rimanenti, introducendo in più uno stato per permettere analisi in termini di metriche di processo.

Si tenga presente, inoltre, che tutti gli automi a stati finiti descritti in questo capitolo sono stati integrati all'interno della specifica SLANG, come riisulterà più evidente nel Cap. 9 e come descritto nella Appendice A.

8.1.1 L'automa del Rapporto di Anomalia

La figura 8.1 mostra l'automa originariamente presente nel Manuale di Qualità. Osservando tale automa, si può notare come sia impossibile annullare la realizzazione della soluzione di un'anomalia, una volta che questa sia

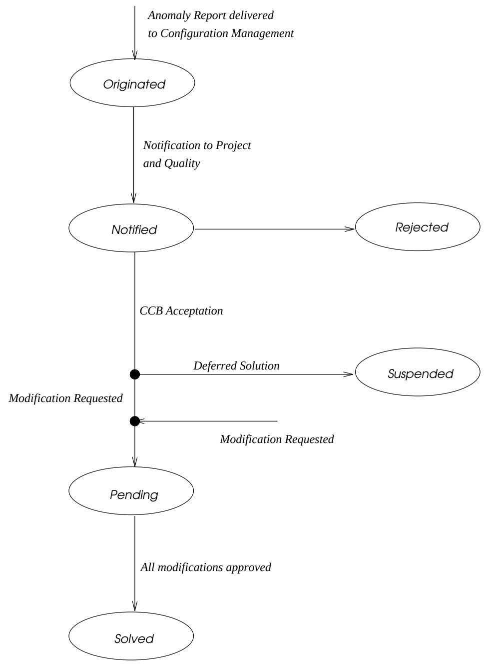

Figura 8.1: La versione originale dell'automa che descrive un AR.

iniziata. Questo aspetto è invece abbastanza comune all'interno del processo di sviluppo di software, in quanto la cancellazione della realizzazione di un'anomalia può avvenire per molti motivi, che includono la decisione di abbandonare una parte di progetto che conteneva l'anomalia, problemi di risorse, cessazione di collaborazione con terze parti e così via.

Pertanto l'aspetto della cancellazione di un cambiamento è un'aspetto critico all'interno della gestione del processo software: a confermare ulteriormente questa affermazione, in letteratura è presente un esempio di processo software [K$^+$90], il cui scopo è permettere di confrontare modellazioni eseguite con vari linguaggi, che richiede proprio la gestione di un cambiamento ad un componente software, includendo la possibilità di annullare tale cambiamento una volta attivato.

Abbiamo fatto presente questo fatto all'esperto, proponendo una nostra soluzione: questa è stata elaborata e definita come in Fig. 8.2.

In tale automa appaiono due nuovi stati:

- **ToDefine.** L'anomalia assume questo stato quando è necessario ac-

Figura 8.2: Una prima versione aggiornata dell'automa per gli AR.

quisire informazioni addizionali, da parte del CCB, per poter decidere se attivare o meno un processo di soluzione dell'anomalia.

- **PartiallySolved.** L'anomalia passa in questo stato quando in archivio di configurazione vi è almeno una delle RIMO associate all'anomalia che sia stata eseguita e revisionata positivamente.

Si noti che lo stato di 'Rejected' svolge anche le veci di stato in cui si termina la soluzione di un'anomalia in seguito ad una cancellazione.

Giunti a questo punto si è ritenuto che la soluzione elaborata non fosse ancora soddisfacente, poiché alcuni aspetti non venivano evidenziati adeguatamente ed altri erano ambigui, come ad esempio l'equivalenza tra due eventi logicamente separati come il rifiuto di un anomalia e la sua cancellazione.

Per questa ragione si è proposto un nuovo automa, mostrato in Fig. 8.3. Si noti come la complessità degli automi cresca con le modifiche: ciò è sintomatico del fatto che la complessità intrinseca del processo software genera descrizioni a loro volta tanto più complesse quanto più si cerca di scendere con il livello di dettaglio.

In questo nuovo automa sono stati distinti gli stati di 'Rejected' e **Cancelled**, assegnando solo a quest'ultimo il significato di anomalia cancellata durante il processo di soluzione.

In aggiunta, è stato inserito uno stato **Refused** nel quale viene fatta transitare l'anomalia quando il modulo del Rapporto di Anomalia è scorretto dal punto di vista formale: tale stato verrà utilizzato all'interno di analisi metriche sul processo.

Inoltre, poiché RIMO ed AR sono interdipendenti, si è cercato di mettere in collegamento i due automi, numerando gli archi corrispondenti ad eventi correlati in maniera omologa su ambedue i diagrammi.

8.1.2 L'automa della Richiesta di Modifica

Nella documentazione a noi inizialmente fornita non vi erano cenni ad un automa per la gestione delle RIMO, analogo a quello già esaminato per la gestione delle anomalie; tuttavia, un colloquio con l'esperto di processo, Ing. Maurizio Loi, ha mostrato che un tale automa in realtà già esisteva ed era utilizzato nella consuetudine produttiva, benché non ancora integrato all'interno della documentazione ufficiale. Tale automa è mostrato in Fig. 8.4.

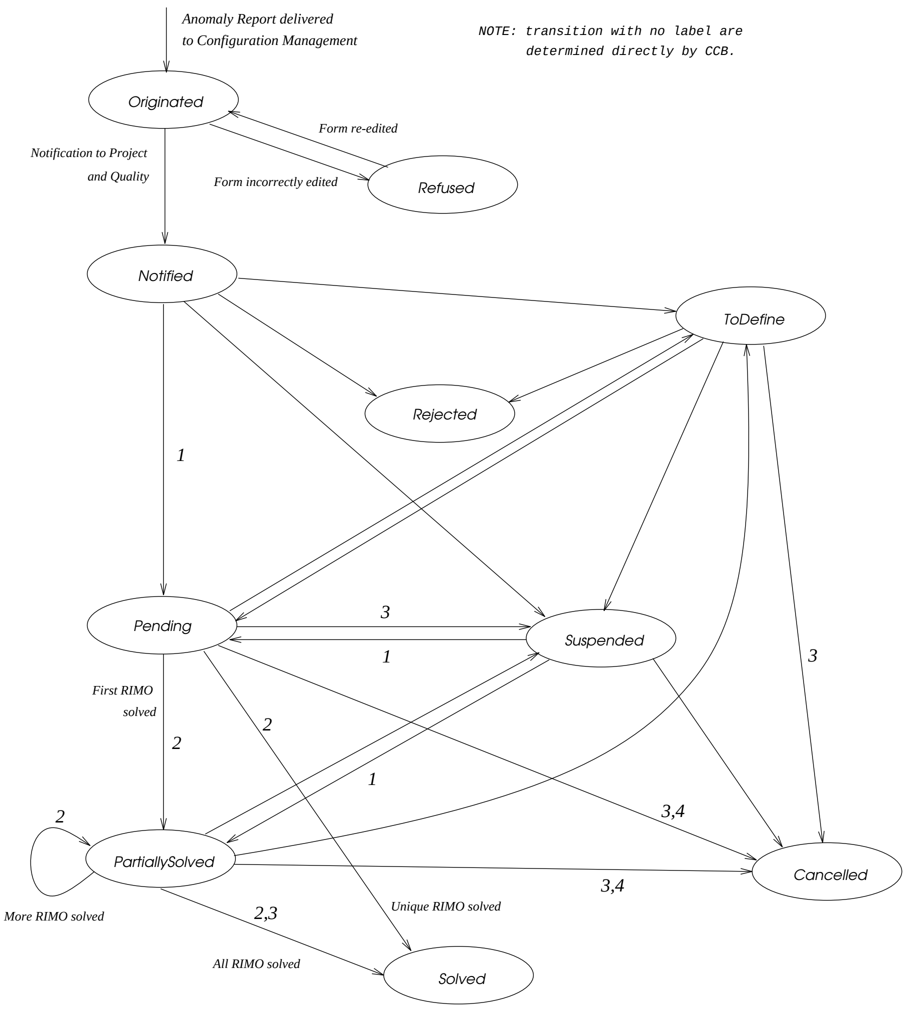

Figura 8.3: La versione definitiva dell'automa per gli AR.

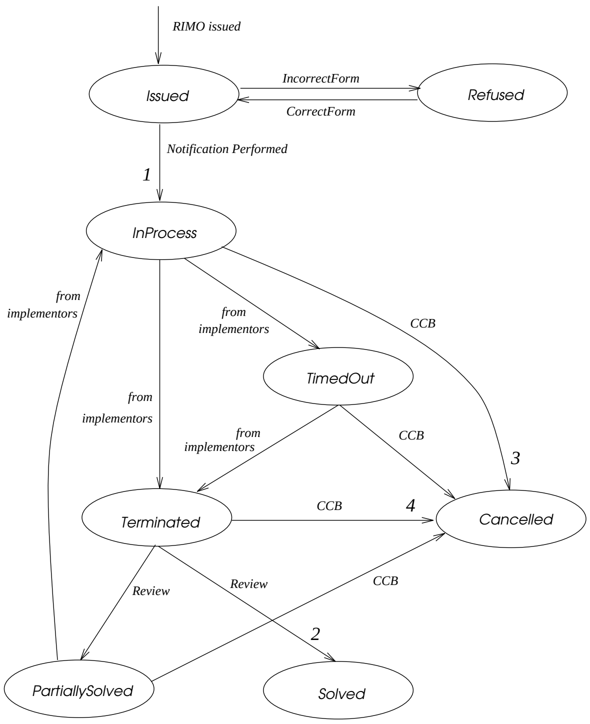

Figura 8.4: La prima versione dell'automa relativo alla gestione delle RIMO.

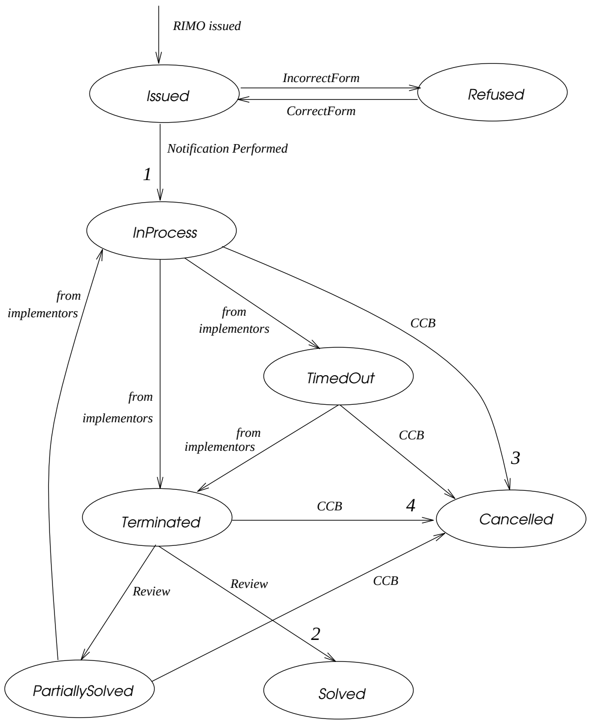

Figura 8.5: La versione definitiva dell'automa per la gestione delle RIMO.

Anche questo automa è stato giudicato inadatto a rappresentare fedelmente la sequenza di passi operativi attuabili all'interno del processo reale, per cui è stato proposto ed adottato l'automa esteso come in Fig. 8.5.

Si noti che in quest'ultimo automa lo stato che in Fig. 8.4 era stato chiamato 'Effectuated' è stato sdoppiato nei due stati 'InProcess' e 'Terminated', per poter effettuare un controllo ulteriore sulla situazione della RIMO. Inoltre, anche in questo caso è stato aggiunto uno stato 'Refused' per la gestione delle metriche.

La descrizione degli stati è la seguente:

- **Issued.** La Richiesta di Modifica è stata generata dal CCB dopo l'esame di un'anomalia.

- **Refused.** La RIMO consegnata al SGMR presenta aspetti scorretti dal punto di vista formale e viene rimandata all'estensore per le dovute correzioni.

- **InProcess.** La RIMO (eventualmente corretta) è stata notificata alla persona che dovrà implementarla.
- **TimedOut.** La RIMO in corso di implementazione ha oltrepassato la scadenza massima consentita per la sua soluzione.
- **Terminated.** L'esecutore della RIMO ha deciso che le modifiche richieste sono state implementate. La RIMO dovrà ora essere revisionata.
- **Cancelled.** Il CCB ha deciso, in seguito a segnalazioni da parte dei tecnici e previo esame delle anomalie associate, che l'esecuzione delle modifiche associate alla RIMO vanno cancellate, qualunque sia lo stato attuale della RIMO.
- **PartiallySolved.** La revisione ha evidenziato che solo alcune delle modifiche contenute nella RIMO sono state effettivamente ed efficacemente implementate. Pertanto, la RIMO viene rimandata all'esecutore.
- **Solved.** La revisione ha evidenziato che la RIMO è stata completamente ed efficacemente implementata. Pertanto la gestione della RIMO può considerarsi terminata.

Si noti la numerazione degli archi, omologa di quella vista per la gestione dei Rapporti di Anomalia.

8.1.3 Gli automi per gli elementi di configurazione

Così come per le Richieste di Modifica, anche per gli elementi in controllo configurazione inizialmente non avevamo alcuna informazione descritta nel Manuale di Qualità. In realtà ci è stata fornita in seguito una istruzione operativa molto dettagliata relativa allo strumento utilizzato per il controllo configurazione, la quale recava anche alcuni automi differenziati secondo le tipologie indicate nella Sez. 8.1. Esamineremo ora la versione modificata, che ha in sostanza introdotto lo stato 'Refused' per il controllo sul numero di elementi rifiutati dal SGMR ed uniformato gli stati iniziali e terminali degli automi.

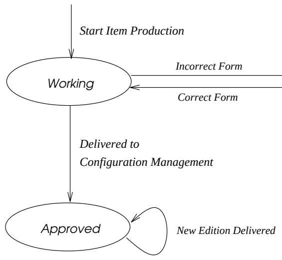

Figura 8.6: L'automa per la gestione dei documenti non soggetti a verifica e validazione.

Automa per i documenti non soggetti a verifica e validazione

L'automa relativo è in Fig. 8.6. Gli stati hanno il seguente significato:

- **Working.** Il documento ha per default questo stato durante la sua costruzione. In realtà è uno stato fittizio in quanto, non essendo depositato in controllo configurazione, non vi è alcun controllo su di esso.
- **Refused.** Il documento è rifiutato dal SGMR poiché formalmente non corretto, e viene rimandato all'estensore per le necessarie correzioni.
- **Approved.** Poiché non è necessaria revisione sul documento, una volta passato il controllo formale dal SGMR, il documento transita direttamente in questo stato.
- **Released.** Alcuni documenti non soggetti a verifica e validazione vengono richiesti dal committente: quando ciò accade, il documento transita in questo stato.

Automa per i documenti soggetti a verifica e validazione

L'automa è in Fig. 8.7 ed è un'estensione del precedente. In aggiunta sono presenti i seguenti stati:

- **Draft.** In questo stato il documento è in uno stato ritenuto consistente dall'estensore e conseguentemente è stato consegnato al SGMR, in attesa di essere sottoposto a revisione.