Ringraziamenti

Un primo, immenso e doveroso ringraziamento deve essere rivolto ai miei genitori che mi hanno sostenuto in questi anni di università permettendomi di laurearmi nonostante tutte le difficoltà che ho incontrato.

Ringrazio Luca (Easy) Messori per tutto l'aiuto che mi ha formito durante la realizzazione di questa tesi; grazie ai miei amici e compagni di università con cui ho condiviso gioie e dolori della vita universitaria.

Ringrazio Patrizia che ha illuminato la via al mio spirito tormentato permettendogli di trovare la strada nella tempesta della vita.

Un grazie di cuore a tutti gli amici di Mirandola per le belle serate passate in compagnia, in particolare alla "Squadra 17".

Un grazie particolare a Daniele (Il Nero) per le pause di studio e per le informazioni sui mangaka.

Inoltre desidero ringraziare: Go Nagai, Rumiko Takahashi, Tsukasa Hojo, Koshi Rukudo, Toshihiko Sato, Akiyoshi Sakai, Kazuo Komatsubara, Tadao Nagahama, Ken Ishikawa, Yu Yamamoto, Tomohatsu Tanaka, Hajime Yata, Kunio Okawara, Mitsuteru, Tadao Nagahama, Yoshitake Suzuki, Yoshiyuki Tomino, Yoshikata Nitta, Osamu Kobayashi non solo per la grande influenza che hanno avuto nella mia infanzia, ma sopratutto per avermi fatto innamorare dell'affascinante cultura giapponese.

Ringrazio Cristina, Gabry, Paolo, Lele e tutti gli altri amici della Compagnia che mi hanno sempre aiutato e con cui ho passato momenti bellissimi.

Un affettuoso ringraziamento alle mie cugine Cristina e Paola per la loro straordinaria amicizia e per tutto l'affetto che mi hanno dato in questi anni. Grazie ai miei zii, in particolare a Gino, Elio e Pasquina, Maria e Severina, Adelino.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Internet è in continua crescita; oggi siamo in presenza di varie decine di milioni di utenti sparsi in tutto il mondo che vi si connettono regolarmente. Con il termine *utente* non ci si riferisce solamente ad una persona che utilizza il proprio PC personale, ma anche a reti locali (per esempio reti aziendali, reti della pubblica amministrazione, reti bancarie, reti che servono singoli edifici, etc.); tali reti contengono **informazioni** e forniscono **servizi**. Queste caratteristiche le rendono bersagli appetibili di malintenzionati interessati a reperire informazioni (si pensi, ad esempio, allo spionaggio industruale oppure ai dati contenuti all'interno della rete della pubblica amministrazione) oppure ad impedire l'erogazione di determinati servizi. E' quindi evidente come al giorno d'oggi nessun amministratore di rete possa permettersi di mettere in secondo piano il problema della sicurezza della rete che amministra, né può scegliere a caso gli strumenti di difesa da attivare. Questi ultimi devono essere identificati in base al tipo di rete locale, alla sua dimensione, al numero ed alle tipologie di utenti che giornalmente si connettono ad essa, al tipo di informazioni che la rete custodisce, etc.

Vari sono gli strumenti che oggi sono presenti sul mercato per creare un buon sistema di difesa (sia hardware che software) e prevedono vari gradi di interazione con l'utente. I sistemi di difesa possono prevedere solamente azioni di rilevamento di attività considerata insolita e sospetta oppure possono anche intraprendere azioni nei confronti degli attaccanti (come ad esempio inviargli messaggi di posta elettronica). L'amministratore della sicurezza può scegliere di adottare strumenti già configurati in modo standard, in grado di proteggere in modo accettabile diverse tipologie di reti locali, oppure utilizzare mezzi adattabili che devono essere configurati ad hoc in base

alla struttura della lan, ma che ne garantiscono una maggiore protezione. Inoltre tali sistemi di sicurezza, qualsiasi essi siano, dovranno essere mantenuti costantemente aggiornati poiché, se da un lato si assiste quotidianamente a progressi nell'ambito della sicurezza, dall'altro anche i sistemi di attacco hanno un'evoluzione altrettanto veloce.

Come è facile immaginare, il problema risulta essere molto complesso e ricco si sfaccettature, ma nonostante questo esiste una necessità che è comune a tutti i sistemi di difesa: la tempestività d'azione. Minore è il tempo impiegato per rilevare un'attività sospetta e per segnalarla all'amministratore della sicurezza, maggiori risultano le possibilità di bloccare sul nascere l'attacco riducendo al minimo i danni subiti, ed aumentare la probabilità di poter ripristinare tutte le funzionalità antecedenti all'attacco in breve tempo e senza perdere dati.

1.1 Obbiettivi

L'obbiettivo di questa tesi è quello di modificare uno strumento di intrusion detection già presente sul mercato (Prelude IDS) rendendolo in grado di sfruttare tutte le potenzialità fornite da un sistema esperto (CLIPS) al fine di creare un Intelligent IDS (IIDS) capace di riconoscere in modo automatico vari tipi di attacchi e di attivare le migliori contromisure al fine di annullarli.

1.2 Struttura della tesi

Nel capitolo 2 verranno analizzati i sistemi esperti presenti oggi sul mercato per poi passare, nel capitolo 3, ad una trattazione più dettagliata del sistema esperto utilizzato per la realizzazione di SLUNP; in particolare, verrà spiegato in cosa consiste un sistema esperto e come è in grado di seguire processi decisionali complessi.

Il capitolo 4 presenta nel dettaglio il sistema Prelude IDS descrivendone il funzionamento ed i componenti fondamentali. Nel capitolo 5 verrà descritto il lavoro di progettazione teorica dell'Intelligent IDS realizzato, chiamato S.L.U.N.P. (Synthetic Lifeform Used for Network Pacekeeping), e nel capitolo 6 verrà descritto quanto realizzato sperimentalmente.

Nel capitolo 7 verranno descritti i risultati sperimentali ottenuti effettuando vari test sul sistema sviluppato, nonché i possibili sviluppi futuri di SLUNP; infine nel capitolo 8 verranno presentati i risultati ottenuti simulando un port scanning.

Capitolo 2

Sistemi esperti

2.1 Introduzione

I linguaggi di programmazione di tipo convenzionale, come Fortran o C, sono progettati ed ottimizzati per la manipolazione di dati di tipo procedurale (es. numeri o vettori). A differenza di questi linguaggi di programmazione, la mente umana non segue affatto un metodo di ragionamento di tipo procedurale: è ingrado di risolvere problemi, anche complessi, utilizzando un approccio simbolico molto astratto, che attualmente non è replicabile utilizzando i linguaggi di programmazione procedurali.

Uno dei risultati della ricerca nell'area dell'intelligenza artificiale è stato lo sviluppo di tecniche che permettono la modellazione delle informazioni lavorando ad un alto livello di astrazione. Queste tecniche sono state inserite in linguaggi di programmazione che permettono di costruire applicazioni che ricostruiscono i processi logici della mente umana in modo molto più preciso rispetto ai linguaggi tradizionali. **I programmi che emulano l'abilità di ragionamento umana in un dominio di problematice ben definito sono chiamati sistemi esperti.** La disponibilità di software di questo tipo ha ridotto in modo evidente lo sforzo e le risorse necessari nella creazione di programmi che riproducono i processi logici umani.

La programmazione basata su regole è la tecnica maggiormente utilizzata per lo sviluppo di sistemi esperti. In questo paradigma di programmazione, le regole sono utilizzate per fare in modo di eseguire determinate azioni qualora si presentino determinate situazioni. Una regola è composta da due parti fonsamentali:

- **IF portion (LHS):** contiene una serie di modelli che specificano i fatti[1] che causano l'applicabilità della regola. Il processo che permette di effettuare il match fra i modelli delle regole e le conoscenze[2] è chiamato pattern matching. Il sistema esperto fornisce un meccanismo, chiamato motore inferenziale, che automaticamente effettua il match fra fratti e modelli e determina quali regole sono applicabili; in particolare non bisogna pensare che questo meccanismo sia effettuato seguendo il paradigma procedurale. Il pattern matching viene effettuato ogni volta che vengono effettuati cambiamenti sui dati utilizzando algoritmi sofisticati, primo fra tutti quello denominato *Algoritmo Rete*.
- **THEN portion (RHS):** contiene il set di azioni da eseguire quando la regola è applicabile. Le azioni delle regole applicabili sono eseguite non appena il motore inferenziale viene istruito ad iniziare l'esecuzione. Questo seleziona una regola, dopo di ché le azioni di tale regola presenti nella then portion vengolo eseguite; il motore inferenziale seleziona quindi un'altra regola permettendo l'esecuzione delle sue istruzioni. Questo processo si ripete fin tanto che ci sono regole applicabili.

Come detto, il motore inferenziale permette l'esecuzione delle azioni presenti nella then portion di una regola applicabile; queste azioni potrebbero andare a modificare le conoscenze del sistema esperto, andando ad operare su un database interno al sistema stesso chiamato Base delle Conoscenze. Questo database viene utilizzato per simulare i ricordi e le esperienze della mente umana e per questo è una delle componenti fondamentali di qualsiasi sistema esperto.

2.1.1 Regole ed Algoritmo Rete

Questo algoritmo, il cui nome vuole significare "*algoritmo basato sulla rete*", permette ai sistemi esperti basati su fatti e regole di essere molto efficienti, sia in termini di peso computazionale, sia in termini di tempo di esecuzione, inteso come il tempo necessario per prendere una decisione sulla base delle conoscenze a disposizione del sistema stesso. Per capire fino in fondo quanto questo algoritmo sia al contempo concettualmente semplice e straordinariamente potente si deve innanzitutto capire che cosa si intende quando si parla di "regola di un sistema espeto". A tal fine con-

[1] Fatto: quando si parla di sistemi esperti questo termine deve essere inteso come dato.
[2] Conoscenza: anche questo termine deve essere inteso come dato.

sideriamo il seguente esempio che mostra come venga definita una regola secondo la sintassi del linguaggio CLIPS:

```
(defrule scadenza-prestiti
  (libro (nome ?x) (stato-prestito scaduto)
                           (prestato-a ?y))
  (prestato-a (nome ?y) (indirizzo ?z))
=>
  (invia-avviso-scadenza ?x ?y ?z))
```

Questo semplice esempio permette di distinguere in maniera chiara le due distinte parti di cui si componogo le regole dei sistemi esperti; in particolare, la parte if fa riferimento a due dati presenti all'interno della base delle conoscenze:

- libro: modella i dati che il sistema esperto deve possedere per indicare se il prestito di un libro sia scaduto o meno. Questa conoscenza è a sua volta composta da tre sottodati (o *slot* come vengono definiti in CLIPS):
  1. nome: nella variabile x (in CLIPS qualsiasi cosa che segua il carattere ? è considerato il nome di una variabile) viene inserito il nome del libro preso in considerazione
  2. stato-prestito: questo campo indica espressamente che il prestito deve esserere scaduto affiché la regola presentata sia applicabile
  3. prestato-a: nella variabile y viene inserito il nome della persona a cui è stato prestato il libro in esame
- prestato-a: modella i dati che il sistema esperto deve possedere per identificare la persona a cui il libro in esame è stato prestato. Questo dato è composto da due sottodati:
  1. nome: la variabile y contiene il nome della persona a cui il libro x è stato prestato
  2. indirizzo: la variabile z contiene l'indirizzo della persona y

Nella parte then della regola viene specificata una sola azione da intraprendere qualora la regola sia applicabile; tale azione consiste nell'inviare un avviso di scadenza a y, all'indirizzo z, riguardante il libro x.

Come viente interpretata la regola sopradescritta dal motore inferenziale? Esso non farà altro che confrontare le condizioni presenti nella parte if della regola con i fatti presenti all'interno della base delle conoscenze ed eseguirà le azioni presenti nella parte then tante volte quante la regola risulta applicabile. Per chiarire meglio questo processo consideriamo un esempio numerico: supponiamo che vi siano dieci libri il cui prestito risulta essere scaduto; supponiamo inoltre che cinque libri siano stati prestati a Eikichi Onizuka e cinque a Ataru Moroboshi. La prima condizione che deve essere soddisfatta risulta essere quella vincolante e cioè che il prestito del libro sia scaduto: selezionati tutti i libri che rispettano questa condizione si passa ad analizzare ogni singolo prestito in modo da reperire il nome della persona a cui il libro è stato prestato per poter poi risalire da questo al suo indirizzo.

Come già accennato in precedenza, in una tipica applicazione sviluppata utilizzando un sistema esperto sono presenti una serie di regole che non subiscono modifiche durante i processi logici del sistema, al contrario invece, i dati presenti nella base delle conoscenze sono oggetto di continue modifiche che comprendono inserimenti, modifiche e cancellazioni. Non bisogna comunque pensare che queste modifiche rappresentino l'attività principale delle azioni presenti all'interno delle regole; esse infatti costituiscono solamente una percentuale molto bassa di tutte le azioni che vengono eseguite per ogni unità di tempo.

Verrà ora mostrato come i sistemi esperti riescano a gestire tutte le regole su cui si basano i loro ragionamenti logici e come possano applicarle in modo corretto nonostante i cambiamenti nella base delle conoscenze. Il primo metodo, che risulta essere anche il più semplice, consite nel tenere una lista di tutte le regole presenti nel sistema ed effettuare su di essa un ciclo perpetuo di controllo verificando per ciascuna regola la sezione if per verificare se si possano effettuare i match necessari fra le condizioni presenti ed i dati in possesso del sistema; qualora tutte le condizioni risultino soddisfatte, verranno eseguite tutte le operazioni presenti nella parte then di ciascuna regola esaminata. Come è facile capire questo sistema risulta essere molto inefficente poiché la maggior parte dei test sulla sezione if, effettuati in ogni ciclo di controllo, forniranno gli stessi risultati dati nell'iterazione precedente; ciò è dovuto al fatto che le modifiche ai fatti contenuti nella base delle conoscenze effettuate per ogni unità di tempo risultano essere una percentuale bassa rispetto al totale delle operazioni eseguite. Questo tipo di approccio viene chiamato *rules finding* e la sua complessità computazionale è nell'ordine di $O(R*F^P)$, dove R rappresenta il numero

delle regole presenti nel sistema, F rappresenta il numero di fatti presenti nella base delle conoscenze e P rappresenta il numero medio di pattern per regola.

Un metodo alternativo e di gran lunga più efficente è conosciuto con il nome di **Rete Algorithm** [Rete-Alg]. In questo algoritmo, l'inefficienza descritta in precedenza viene in parte eliminata modificando radicalmente il metodo con cui si effettuano i match fra i vari dati: questo algoritmo permette infatti di tenere traccia dei risultati dei testi effettuati in ogni singolo ciclo di controllo. Ciò permette di testare le condizioni presenti nella parte if delle varie regole solamente sui nuovi dati inseriti nella base delle conoscenze; inoltre queste nuove conoscenze vengono testate solamente rispetto alle condizioni delle regole in cui sembrano essere maggiormente indicative per ottenere l'applicabilità della regola.

Il risultato di questo approccio è dato da un drastico abbattimento della complessità computazionale per orgni iterazione che risulta essere pari a $O(R * F * P)$ e quindi lineare rispetto alla dimensione della base delle conoscenze F [Compl-Rete].

L'algoritmo Rete viene implementato costruendo una rete di nodi, ciascuno dei quali rappresenta uno o più condizioni trovate nella parte if di una regola. I fatti aggiunti o rimossi dalla base delle conoscenze sono processati da questa rete alla cui fine si trovano nodi che rappresentano regole individuali. Quando una serie di conoscenze filtra attraverso tutta la rete fino a raggiungere il fondo significa che esse hanno soddisfatto tutte le condizioni specificate nella parte if di una data regola che può quindi essere applicata. L'applicabilità della regola fa sì che tutte le azioni presenti nella sua parte then vengano eseguite quando la regola viene attivata dal motore inferenziale, sempre ché la rimozione di un qualche dato dalla base delle conoscenze effetuata prima dell'attivazione della regola ne annulli l'applicabilità.

Analizziamo ora la struttura della rete alla base di questo algoritmo. Essa si compone di due tipi principali di nodi: i nodi *one-input* e *two-input*. I primi effettuano test su fatti individuali, mentre i secondi effettuano test fra dati diversi e svolgono le funzioni di raggruppamento dei fattis tessi. Vi sono inoltre sottotipi di queste due classi di nodi, oltre a classi ausiliarie, come ad esempion i nodi terminali situati alla fine della rete. Per rendere il concetto più chiaro consideriamo il seguente esempio, dove vengono prese in considerazione due regole e tre conoscenze:

```
(defrule es1                          (defrule es2
  (x)                                   (x)
```

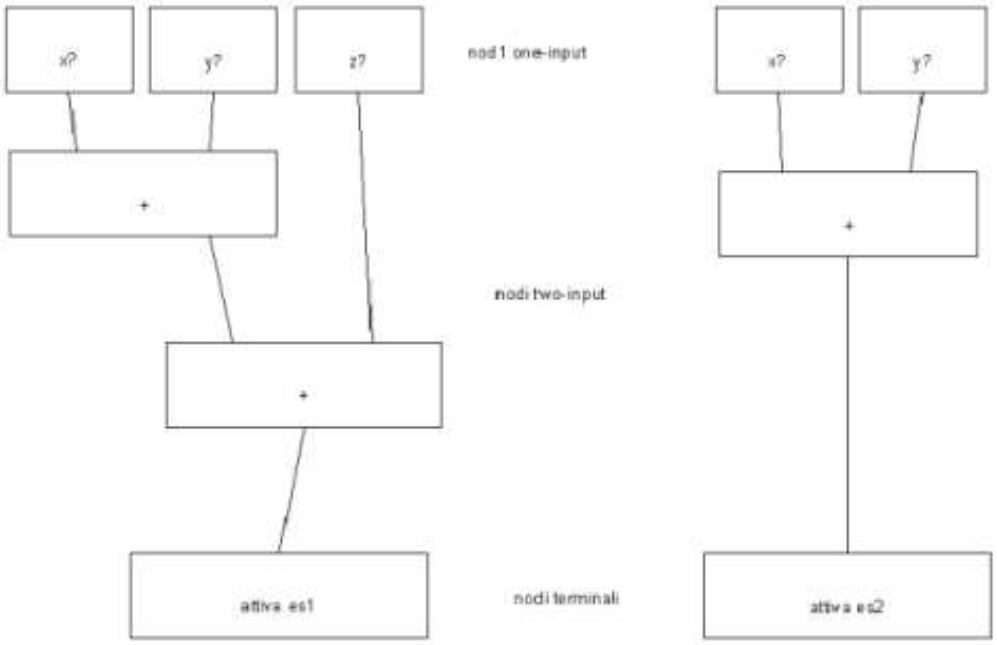

Figura 2.1: Primo esempio di rete costruita utilizzando il Rete Algorithm

```
   (y)                                    (y)
   (z)
=>)                                    =>)
```

La rete che si ottiene da questo esempio è rappresentata in figura 2.1; i nodi marcati "x?". "y?", "z?" testano se un fatto contiene le informazioni specificate, mentre i nodi marcati "+" ricordano tutte le conoscenze e la applicabilità ogni qual volta ricevono informazioni da entrambi i loro inputs (destro e sinistro). Quando nuovi dati vengono inseriti nella base delle conoscenze, queti vengono inseriti nei nodi situati alla testa della rete; ciascun nodo prende il suo input dall'alto, effettua un test specifico e produce un output verso il basso. Se il test fallisce, i nodi one-input non danno alcun output; i nodi two-input, a differenza dei precedenti, devono integrare fatti che raccolgono dal flusso di dti attraverso i loro due rami di input. Anche i processi che avvengono al loro interno sono ben più complessi di quelli presenti nei nodi one-input: bisogna infatti tener presente che ogni singolo dato che raggiunge un ramo di input di uno di questi nodo potrebbe potenzialmente dare luogo all'applicabilità di una regola, per questo motivo i nodi two-input effettuano su ogni singolo

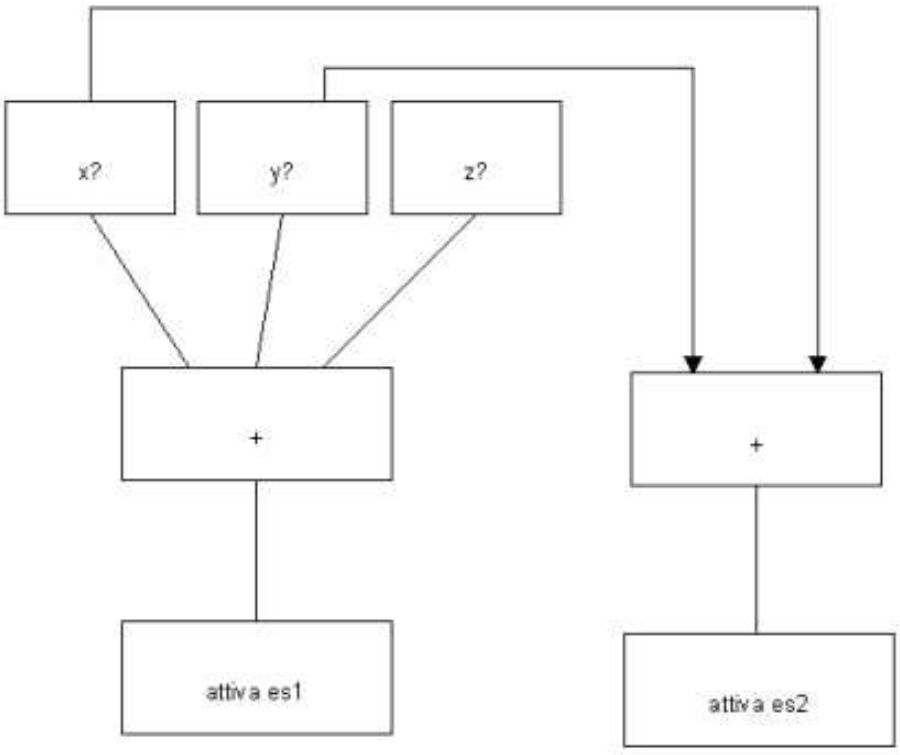

Figura 2.2: Prima ottimizzazione della rete

dato che giunge loro in ingresso da uno dei due rami tutti i possibili match con i dati provenienti dall'altro ramo di input. Inoltre ciascuno di essi deve memorizzare ogni singola conoscenza che lo ha raggiunto suddividendo tale memoria fra parte destra e parte sinistra; proprio tale memoria, come già descritto precedentemente, costituisce la forza di questo algoritmo, consentendo di ridurre in modo drastico la complessità computazionale del problema dell'applicabilità delle regole.

Osservando la rete di nodi costruita in figura 2.1 si può notare che essa può essere suddivisa in due parti: la *pattern network*, costituita dai nodi one-input, e la *join network*, costituita da nodi two-input.

Sinora si è mostrato quali sono i punti di forza del Rete Algorithm, ma nonostante ciò non bisogna pensare che sia perfetto: sono infatti possibili due distinti miglioramenti, basati entrambi sulla necessità di ridurre le ridondanze presenti nella rete descritta nell'esempio precedente.

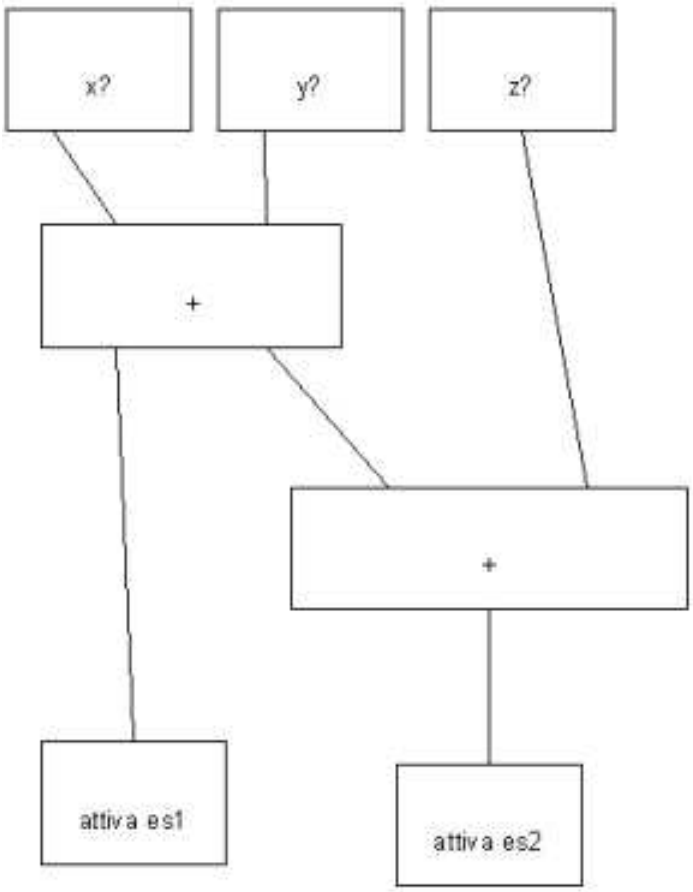

Figura 2.3: Completa ottimizzazione della rete

Il primo consiste nella condivisione dei node della pattern network, come mostrato in figura 2.2, dove la rete originale di figura 2.1 è stata modificata ottenendo una rete composta da cinque nodi one-input di cui, utilizzando il metodo della condivizione, solamente tre sono distinti.

Sebbene questo consenta di eliminare molte ridondanze, tuttavia non permette di costruire una rete del tutto ottimizzata. Come è facile notare infatti vi sono dei nodi appartenenti alla join network che compiono esattamente le stesse funzioni, quindi anche questi due nodi si possono considerare indistinguibili e perciò possono essere fusi, come mostrato in figura 2.3.

La pattern network e la join network rappresentano in totale una rete la cui dimensione è solamente la metà di quella originale; è facile comprendere come questo

tipo di fusione fra nodi consenta di ottenere un grandissimo incremento in termini di prestazioni.

2.2 Classificazione dei sistemi esperti

Dopo aver illustrato le componenti principali di un sistema esperto, passiamo ora ad elencare i principali prodotto disponibili sul mercato, effettuando una semplice classificazione basata sul diverso tipo di licenze di utilizzo concesse all'utente.

2.2.1 Sistemi Esperti Public Domain

2.2.1.1 C.L.I.P.S.

CLIPS (C Language Integrated Production System) è uno strumento per lo sviluppo e la distribuzione di sistemi esperti che mette a disposizione un completo ambiente per la costruzione di sistemi esperti basati su regole e/o oggetti [CLIPS]. Creato nel 1985, CLIPS è oggi largamente usato per applicazioni governative, industriali ed accademiche. Le sue caratteristiche chiave sono:

- Knowledge Representation: CLIPS fornisce uno strumento potente per maneggiare una vasta varietà di conoscenze con il supporto di tre differenti paradigmi di programmazione:

  1. rule-based: la programmazione basata su regole permette di rappresentare le conoscenze in modo euristico, specificando per ogni singola situazione che si può verificare una serie di azioni da intraprendere.
  2. object-oriented: la programmazione orientata ad oggetti permette di modellare sistemi, anche complessi, attraverso componenti modulari che possono essere facilmente riutilizzati per la modellazione di altri sistemi o per creare nuovi componenti.
  3. procedurale: le capacità di programmazione procedurale rese disponibili da CLIPS sono simili a quelle rese disponibili da linguaggi come C, Java, Ada e LISP.

- Portabilità: CLIPS è scritto in C per questioni di portabilità e velocità; questa scelta ha permesso di renderlo portabile su differenti tipi di sistemi operativi

senza necessità di operare modifiche nel codice. Alcuni esempi di sistemi operativi su cui CLIPS è stato testato sono: Windows 95/98/NT, MacOS X e Unix. CLIPS è un sistema che può essere installato in qualsiasi sistema che abbia un compilatore C o C++ di tipo ANSI.

- Sviluppo interattivo: la versione standard di CLIPS fornisce un ambiente di sviluppo interattivo di tipo testuale che include aiuti e suggerimenti per il debug, un manuale on-line ed un editor integrato.

- Verifica/validazione: CLIPS include molti strumenti che supportano la verifica e la validazione di sistemi esperti; questi strumenti includono un supporto per il design modulare e partizionato di una base di conoscenze, uno strumento per la verifica dell'assegnamento statico e dinamico dei valori degli argomenti dei vari campi degli oggetti, un analizzatore della semantica delle regole che determina se è possibile prevenire eventuali inconsistenze che potrebbero generare stati errati capaci di far scattare alcune regole in momenti sbagliati.

2.2.1.2 Sistemi sviluppati sull'architettura CLIPS:

1. **DYNACLIPS** (DYNAmic CLIPS Utilities)Si tratta di una serie di "scatole nere", scambio di dati e tool di agenti per CLIPS 5.1 e 6.0. E' implementato come una serie di librerie che possono essere associate alle due versioni di CLIPS. Gli agenti utilizzano le scatole nere per comunicare con gli altri agenti presenti nella struttura del sistema esperto sviluppato. Ciascun agente può ricevere e spedire conoscenze, regole e comandi. Regole e fatti sono inseriti e cancellati dinamicamente mentre si stanno eseguendo gli agenti. [DYNACLIPS]

2. **AGENT_CLIPS** E' un tool multi-agente per MACINTOSH [MAC]. Copie multiple di CLIPS sono eseguite contemporaneamente su un sistema MACINTOSH. Ciascuna copia (agente) può spedire comandi CLIPS agli altri agenti attivi a run time. AGENT_CLIPS processa i comandi in ingresso automaticamente. Questa è una forma di scambio di conoscenze fra agenti intelligenti che non utilizza un'architettura basata su "scatole nere" come il precedente DYNACLIPS. [AGENT-CLIPS]

3. **FuzzyCLIPS** E' una versione del sistema esperto CLIPS che presenta un'estensione per rappresentare e manipolare la logica fuzzy. Oltre alle normali

funzioni di CLIPS, questo tool permette di utilizzare la logica fuzzy, sia da sola che combinata con la logica "esatta" di CLIPS permettendo di mixare liberamente componenti "normali" con componenti fuzzy in fatti e regole. Il sistema utilizza principalmente due concetti della logica inesatta (fuzzy): *fuzziness* ed *incertezza*. Sono disponibili versioni per sistemi UNIX [UNIX], MACINTOSH e PC. [FUZZY]

4. **wxCLIPS** Fornisce una semplice interfaccia grafica per CLIPS 5.1, CLIPS 6.0 e CLIPS 6.0 con estensione fuzzy [wxCLIPS]. E' disponibile per Sun Open Look, Sun Motif, Linux Open Look [SUN], Windows 3.1, Windows 32bit e Windows 95 binaries [Win].

2.2.1.3 FOCL

Si tratta di un software per lo sviluppo di sistemi esperti e per la costruzione di macchine capaci di apprendere autonomamente; tale software è scritto in Common Lisp; in particolare, nasce come estensione del programma denominato FOIL. Tale estensione è stata resa possibile inserendo in FOCL un componente di apprendimento che riesce ad interpretare le "spiegazioni" (dati) fornite al sistema. FOCL utilizza la logica di Horn Clause per realizzare l'autoapprendimento attraverso esempi e conoscenze di background. Questo sistema esperto include un interprete di regole ed un'interfaccia grafica per la gestione delle conoscenze e delle regole. [FOCL]

2.2.1.4 SOAR

SOAR è un'architettura generale di tipo cognitivo, nata nel 1983, per lo sviluppo di sistemi che presentano un comportamento di tipo cognitivo. [SOAR]

I principi base che hanno guidato lo sviluppo di questo sistema esperto sono:

- Minimizzare il numero di meccanismi architetturali distinti. In SOAR c'è un singolo framework per ciascun componente e sottoproblema, una singola rappresentazione delle conoscenze permanenti, una songola rappresentazione delle conoscenze temporanee (oggetti con attributi e valori), un singolo meccanismo per il raggiungimento degli obbiettivi, un singolo meccanismo di apprendimento.

- Far sì che tutte le decisioni siano prese a run-time attraverso la combinazione di conoscenze rilevanti. In SOAR ogni decisione è basata sulla interpretazione

dei dati sensoriali, sul contenuto della memoria che tiene traccia delle soluzioni dei problemi precedentemente risolti e sulle conoscenze presenti all'interno della memoria permanente del sistema; in poche parole, la sequenza di decisioni non è mai prestabilita, ma risulta il frutto di attività a run-time.

2.2.1.5 OPS5

Fu il primo linguaggio di programmazione basato sull'algoritmo Rete ed il primo linguaggio di intelligenza artificiale che ebbe successo in ambito di applicazioni commerciali, sin da quando il dottor John McDermott implementò un software di configurazione basato su regole per sistemi di computer VAX per la Digital Equipment Corporation. L'applicazione, chiamata R1, fu inizialmente sviluppata in Common Lisp, ma successivamente fu tradotta in OPS5 per ragioni di prestazioni; tale versione prese il nome di XCON. XCON ebbe un successo strepitoso; fu il pioniere di un grandissimo numero di applicazioni sviluppate utilizzando OPS5, molte delle quali create da Paul Haley [OPS5].

2.2.1.6 BABYLON

E' un ambiente di sviluppo per sistemi esperti. Include frames, constrains, un formalismo in stile PROLOG ed un linguaggio descrittivo per applicazioni diagnostiche. E' implementato in Common Lisp ed è stato installato in una vsta gamma di piattaforme hardware. [BABYLON]

2.2.1.7 Jess

Jess è sia un motore basato su regole che un ambiente per gli script scritto interamente nel linguaggio Java nei Sandian National Laboratories in Livermore (CA). Jess fu originariamente ispirato dal sistema CLIPS, ma si è poi evoluto in un sistema completamente distinto dal suo ispiratore sviluppando un suo proprio ambiente dinamico. Utilizzando Jess si possono sviluppare applicativi Java che hanno la capacità di "ragionare" usando le conoscenze che vengono fornite loro.

il vero cuore di Jess rimane comunque ancora compatibile con CLIPS: infatti molti script di Jess sono validi anche nell'ambiente CLIPS e viceversa. Come CLIPS, anche Jess usa l'algoritmo Rete per processare le regole; tale algoritmo si è dimostrato un efficace strumento per risolvere problemi di matching molto complicati [JESS].

2.2.1.8 MOBAL

E' un sistema per lo sviluppo di modelli operazionali di applicazioni in una rappresentazione logica di primo ordine. Integra un metodo di apprendimento manuale, uno automatico per l'acquisizione dei dati, un motore inferenziale ed un tool per la revisione delle conoscenze acquisite. Questo sistema è utilizzabile su Sun SparcStations, SunOS 4.1 ed include un'interfaccia grafica implementata usando Tcl/TK[MOBAL]

2.2.1.9 MIKE

MIKE (Micro Interpreter for Knowledge Engineering) è un ambiente software completo, gratuito e portabile progettato per scopi didattici all'UK's Open University [MIKE]. Include un concatenamento delle regole all'avanti ed all'indietro con strategie per risolvere i conflitti sulle regole definibili dall'utente ed un linguaggio per la rappresentazione dei frame che include ereditarietà e "demoni"[3], con la possibilità da parte dell'utente di definire nuove strategie di ereditarietà. Il sistema fornisce vari sistemi per la tracciabilità delle regole tra cui un'interfaccia grafica che permette di visionare in modo veloce la storia dell'esecuzione di ogni singola regola. MIKE è scritto in Prolog [PROLOG].

2.2.1.10 WindExS

WindExS (Windows Expert System) è un completo sistema esperto che utilizza il concatenamento delle regole all'avanti basato sul sistema Windows. La sua architettura modulare permette all'utente di sostituire nuovi moduli ai precedenti in modo semplice per permettere di incrementare le capacità del sistema. WindExS include un processore che permette di utilizzare regole formulate utilizzando il "linguaggio naturale"; include inoltre un motore inferenziale, un file manager, un'interfaccia grafica, un menager dei messaggi e moduli per la base delle conoscenze [WINDEXS].

2.2.1.11 RT-Expert

E' un sistema esperto shareware che permette ai programmatori che usano il linguaggio C di includere nelle loro applicazioni regole di sistemi esperti. RT-Expert consiste in un compilatore di regole che ne consente la traduzione in codice C; RT-Expert in-

[3]Demone: codice eseguito automaticamente ogni volta che si accede o si modifica un frame specificato

clude inoltre una libreria che implementa il motore di esecuzione delle regole stesse. Questo tool, nella versione MS-DOS, può essere utilizzato con i compilatori Borland Turbo C, Borland C++ e Microsoft C/C++. [RT-Expert]

2.2.2 Sistemi esperti commerciali

2.2.2.1 ACQUIRE

E' sia un sistema di acquisizione di conoscenze che una shell per la creazione di sistemi esperti. E' un ambiente completo per la creazione ed il mantemimento di applicazioni basate su conoscenze. Fornisce una metodologia di tipo step-by-step per la progettazione di sistemi esperti che permette agli sviluppatori di sistemi esperti di progettare nuovi tool molto completi e di migliorare quelli già esistenti. ACQUIRE comprense un approccio strutturato per l'acquisizione delle conoscenze, un modello di acquisizione dei dati basato sul riconoscimento di modelli, rappresentazione delle conoscenze sottoforma di oggetti, produzione di regole e tabelle decisionali. ACQUIRE permette anche di processare le incertezze in modo qualitativo, permette di utilizzare sofisticati processi per la scrittura di vari tipi di rapporti e permette di autodocumentare la base delle conoscenze attraverso un ipertesto. [ACQUIRE]

2.2.2.2 ACTIVATION FRAMEWORK

Questo tool non è una tradizionale shell per sistemi esperti, piuttosto è un tool per creare applicazione per l'interpretazione di dati a real-time. E' utilizzabili su qualsiasi sistema Unix, DOS e Windows.

2.2.2.3 Aion Development System (ADS)

E' utilizzabile su numerose piattaforme, incluse piattaforme DOS, OS/2, Microsoft Windows e VMS. Include una rappresentazione dei dati object-oriented, regole concatenate all'atanti, all'indietro ed in modo bidirezionale nonchè regole che utilizzano il pattern matching, possibilità di generare grafici e di effettuare chiamate provenienti e dirette ad altri linguaggi di programmazione come C o Pascal. [ADS]

2.2.2.4 Angoss Knowledge Seeker

Si tratta di un tool per il data-mining che può essere utilizzato per produrre basi delle conoscenze composte da regole grazie a relazioni inferenziali di causa-effetto

rilevate in un qualsiasi database. [AKS]

2.2.2.5 ART*Enterprise

Si tratta dell'ultimo arrivato nella famiglia degli ambienti di sviluppo di sistemi esperti originati attraverso ART nella metà degli anni '80. ART*Enterprise è un ambiente di sviluppo per applicazioni di grandi dimensioni; incorpora regole, un completo sistema oggetto che include strumenti non presenti nel linguaggio C++ o Smalltalk ed una vasta gamma di classi di oggetti. [ART*]

2.2.2.6 Arty Expert Development Package

E' un sistema esperto che integra una rappresentazione delle conoscenze basata su regole con una rappresentazione basata su frames.

2.2.2.7 DClass

Si tratta di un sistema basato su alberi decisionali usato per applicazioni di carattere industriale. [DClas]

2.2.2.8 LogicTree

E' un sistema per il raggiungimento di decisioni rivolto verso utenti che non sono dei programmatori esperti.

2.2.2.9 COMDALE/C

E' un sistema esperto di tipo real-time progettato per il monitoraggio ed il controllo di processi industriali; ammette richieste per giustificazioni o raccomandazioni e permette azioni di controllo che vengono attuate senza interrompere il processo decisionale. Può operare anche in presenza di incertezze nella base delle conoscenze o nei dati ed è provvisto di un'architettura aperta e di un sistema di ragionamento basato sul tempo. In questo tool sono inoltre compresi un'orientazione object-oriented, un driver di controllo degli interrupt, un sistema capace di raccogliere ed analizzare i dati per creare rapporti di trend ed analisi storiche, un time-scheduler degli eventi, un database di tipo real-time. [COMDALE]

2.2.2.10 COMDALE/X

Si tratta di un sistema esperto consultativo che opera off-line; effettua domande agli utenti per ottenere i dati necessari al raggiungimento delle decisioni. COMDALE/C e COMDALE/X sono due software che sono inclusi in un unico pacchetto per lo sviluppo di sistemi esperti real-time. In particolare, COMDALE/X ha la capacità di fondere documenti in formato ipertestuale con le capacità di ragionamento di un sistema esperto per crare manuali ipertestuali "esperti" che forniscono informazioni e generano avvisi attraverso un'interfaccia user-friendly. [COMDALE]

2.2.2.11 ProcessVision

ProcessVision è un sistema di monitoraggio e controllo real-time che fornisce un'interfaccia grafica attraverso la quale l'operatore può controllare vari processi di produzione industriale tramite i dati forniti da sensori. [COMDALE]

2.2.2.12 C-PRS

L'obbiettivo di C-PRS (Procedural Reasoning System in C) è quello di rappresentare ed eseguire procedure operative. Permette all'utente di esprimere e rappresentare le sequenze condizionali di operazioni molto complesse e di assicuare la loro esecuzione in real-time mentre vengono incluse nell'ampiente dell'applicazione. C-PRS è utile nell'implementazione di procedure di controllo e di supervisione di processi di vario tipo. La tecnologia PRS è stata impiegata per il controllo di svariati processi industriali, come il controllo di sistemi di telecomunicazioni od il controllo di sistemi utilizzati sullo space shuttle della NASA. La tecnologia PRS su inizialmente sviluppata all'Artificial Intelligence Center dello Standord Research Institute (Menlo Park, California).

C-PRS è disponibile in varie versioni per diversi sistemi operativi, come SPARC, DECstation, Sony News, Hewlett Packard, VxWorks. [DClas]

2.2.2.13 CPR

CPR (Casa-based Problem Resolution) è una libreria di C++ che opera insieme a Help!CPR, un'authoriting application per conoscenze ed helpdesk. [CPR]

2.2.2.14 CxPERT

E' una shell che produce codice C libero da royalty.

2.2.2.15 The Easy Reasoner (TM)

Si tratta di un tool di tipo Case-Based Retrieval (CBR) che fornisce una memoria associativa adattabile; riconosce casi simili presenti in memoria ed utilizza questa sua capacità per generare nuovi dati; estende il concetto di QBE (Query-by-Examples) fornendo un nuovo tipo di funzionalità chiamato Query-by-Similarity(TM) (QBS); indicizza database esistenti; massimizza le informazioni ed al contempo minimizza la complessità; fornisce filtri automatici che permettono di semplificare gli alberi decisionali; riorganizza le classificazioni dei dati ustando il concetto di similitudine; identifica in modo efficente casi simili presenti in database di vaste dimensioni; supporta alberi decisionali multipli per l'indicizzazione di database. Classifica i nuovi dati utilizzando tutti gli alberi decisionali; classifica in modo automatico o tramite interazione con l'utente nuovi dati; gestisce automaticamente spazi di similitudine N-dimensionali; aggira problemi grammaticali utilizzando una grammatica di tipo N-M. [Easy]

2.2.2.16 Esteem

E' un tool dotato di capacità di ragionamento per Windows basate sul concetto case-based. Integra ragionamenti di tipo case-based con ragionamenti basati su regole. [DClas]

2.2.2.17 EXSYS Professional

E' disponibile per sistemi MS-DOS, Windows, Macintosh, SunOS, Solaris, Unix e Vax. Supporta regole concatenate in avanti ed all'indietro, la programmazione di tipo lineare, la logica fuzzy, le reti neurali ed ha un'interfaccia SQL. [Exsys]

2.2.2.18 FLEX

Si tratta di un sistema esperto di tipo ibrido disponibile in varie versioni riferite a vari tipi di piattaforme hardware. FLEX include frames, procedure e regole integrate all'interno di un ambiente di programmazione logica. Supporta concatenamenti all'avanti ed all'indietro, ereditarietà multiple ed un sistema automatico di domanda e risposta. Regole, frames e domande sono descritte un linguaggio chiamato Enclish-like Knowledge Specification Language (KSL) che permette lo sviluppo di una base delle conoscenze di tipo easy-to-read ed easy-to-mantain. FLEX è implementato in

Prolog ed ha la possibilità di interfacciarsi con questo linguaggio di programmazione. [FLEX]

2.2.2.19 GBB

Generic BlackBoard framework formisce:

- un compilatore ad alte prestazioni basato su un database di black box, associato ad una libreria di runtime, ad alte prestazioni, che supporta algoritmi di ricerca multidimensionale di tipo range-searching per incrementare l'efficienza del ritrovamento, basato sul concetto di prossimità, di oggetti presenti nelle scatole nere.
- un linguaggio di rappresentazione KS
- una shell per il controllo generico ed utilità per la gestione di scheduler
- un'interfaccia grafica interattiva per monitorare ed esaminare le scatole nere e per controllare i vari componenti

GBB è disponibile per MS-DOS, Windows, Mac, Unix workstations (Sun, HP/Apollo, IBM, DEC, Silicon Graphics). [GBB]

2.2.2.20 GURU

E' un ambiente per lo sviluppo di sistemi esperti ed un RDBMS che offre una vasta varietà di tools per l'analisi di informazioni combinati con capacità di tipo knowledge-based come concatenamento all'avanti, all'indietro e di tipo misto, variabili multivalore e ragionamenti di tipo fuzzy. [DClas]

2.2.2.21 HUGIN System

E' un pacchetto software per la costruzione di modelli di base per sistemi esperti caratterizzati da un tipo di ereditarietà incerta. Contiene un sistema deduttivo facile da utilizzare ed estremamente portabile, applicabile a reti complesse che utilizzano relazioni di tipo causa-effetto o di tipo incerto. [HUGIN]

2.2.2.22 Icarus

Si tratta di un tool per lo sviluppo di sistemi esperti per PC. Include link a file Lotus e dBASE, un concatenamento di tipo all'avanti ed all'indietro e fattori di confidenza di tipo Bayesian.

2.2.2.23 ILOG RULES

E' un motore inferenziale ad alte prestazini basato su regole scritto in C++ e fornito sotto forma di libreria C++. Fondamentalmente viene utilizzato per estendere OPS/5 in vari modi, tra cui permettere di compilare regole di sistemi esperti in modo da tradurle in linguaggio C/C++. Inoltre il codice C/C++ può essere direttamente utilizzato all'interno delle condizioni e delle azioni delle regole. [ILOG]

2.2.2.24 KDS

Si tratta di un sistema case-based che produce regole partendo da singoli casi specifici.

2.2.2.25 MailBot

MailBot è un agente mail per Microsoft Mail. Fornisce servizi di filtraggio ed inoltro di mail; può essere utilizzato come un sistema esperto per servizi di mail. Il front-end traduce le regole che l'utente inserisce come input in un linguaggio che risulta essere una variante di Prolog. Esiste inoltre un'API per scrivere le azioni da intraprendere in VisualBasic. [MailBot]

2.2.2.26 MEM-1

MEM-1 è un linguaggio basato su LISP che aiuta nello sviluppo di sistemi di ragionamento case-based; fornisce strumenti per definire strutture che rappresentano singoli casi, per creare nuovi casi, per dividere casi in sotto-casi, per adattare soluzioni e per definire regole di adattamento procedurale. [DClas]

2.2.2.27 Object Management Workbench* (OMW)

OMW è il primo tool di analisi e sviluppo di tipo object-oriented che incorpora direttamente diagrammi executable e regole buisness. OMW lavora con Kappa (un ambiente visuale ed integrato per lo sviluppo di applicazione client/server). Le

applicazioni Kappa sono sviluppate su Unix e possono essere utilizzate sia in Unix che in Windows. [DClas]

2.2.2.28 OPS83

OPS83 fu sviluppato dai creatori di OPS5 come suo successore. E' scritto in C e può essere integrato con esso. In particolare, OPS83 fu il primo tool generato da OPS5 ad avere questa possibilità di integrazione. OPS83 supporta la nuova struttura di controllo GFC (Generalized Foward Chaining) che permette alle regole di essere maggiormente espressive; una regola che appartiene alla struttura GFC può rimpiazzare molte regole tradizionali. [OPS83]

2.2.2.29 RAL

RAL (Rule-extended Algorithmic Language) è un sistema esperto ad alte prestazioni basato sul linguaggio C. Fu sviluppato per permettere l'integrazione di regole ed oggetti nei programmi C senza bisogno di aggiungere del codice specifico al file sorgente. Le regole di RAL possono operare direttamente sui tipi di dato usati nel codice C; RAL riesce ad identificare e "capire" le dichiarazioni dei tipi, i prototipi delle funzioni ecc. Le espressioni C, gli statements, le macro, e tutte le altre funzioni possono essere implementate direttamente nelle regole. RAL ha un'architettura aperta che gli permette di essere usato con qualsiasi GUI builder, con qualsiasi libreria per database o qualsivoglia altra libreria che può essere utilizzata in C. Per ragioni di efficenza, le regole di RAL sono compilate direttamente in codice C. Inoltre va rilevato che Charles Forgy, l'inventore dell'algoritmo Rete (vedi sezione 2.1.1) ha sviluppato un algoritmo di match migliore chiamato Rete II che è stato utilizzato in RAL.

Vi è poi una versione speciale, ciamata RAL/RT, che viene utilizzata in quelle situazioni che richiedono la presenza di un sistema esperto real-time.

2.2.2.30 Rete++

Supporta sia un concatenamento all'avanti che all'indietro. Utilizzando REte++ il programmatore può sviluppare una gerarchia degli oggetti e delle istanze; manipolarli ed accedervi usando sia istruzioni C++ o C, sia la sitassi standard di un sistema esperto basato su regole. Per inserire o modificare dati che sono legati alle regole viene creata un'istanza C++, dopo di ché gli inserimenti e le modifiche vengono effettuate nell'area dati di quell'istanza. Rete++ genera in automatico una

tassionomia di una classe C++. Le componenti C++ delle applicazioni Rete++ utilizzano queste classi generate in automatico in modo diretto oppure trasformandole in sottoclassi se necessario. Il motore inferenziale di Rete++ considera automaticamente ogni istanza di una classe generata (o delle sottoclassi) nelle condizioni di match presenti nelle regole. I tipi di dato forniti da Rete++ che vengono utilizzati in C++ permettono una rappresentazione delle conoscenze maggiormente flessibile, nonché la possibilità di ottenere un processo di ragionamento automatico utilizzando il linguaggio C++ senza la necessità di effettuare chiamate a funzioni esterne. Rete++ monitorizza automaticamente le modifice negli oggetti C++ senza avere la necessità di utilizzare chiamate esplicite a funzioni esterne. Vi sono inoltre moduli che permettono a Rete++ di utilizzare ragionamenti case-based e di interfacciarsi con dei database. [Rete++]

Capitolo 3

Il sistema esperto C.L.I.P.S.

3.1 Evoluzione di CLIPS

Le origini di CLIPS risalgono al 1984 ne Johnson Space Center della NASA. Fino ad allora l'Artificial Intelligence Section aveva sviluppato circa una dozzina di prototipi di applicazioni basate sulla filosofia dei sistemi esperti usando lo stato dell'arte del software e dell'hardware. Comunque, a discapito delle numerose dimostrazioni del potenziale dei sistemi esperti, solamente pochissime applicazioni di quel tipo furono utilizzate regolarmente nelle attività del centro spaziale. Questo fallimento nell'integrare la tecnologia dei sistemi esperti nei sistemi di computazione della NASA può essere largamente localizzata nell'utilizzo di LISP come linguaggio base per lo sviluppo di tutti i pacchetti software di sistemi esperti. In particolare, tre furono i problemi che impedirono l'uso di sistemi esperti per le applicazioni NASA:

- La scarsa portabilità di LISP su una vasta gamma di computer "convenzionali"
- L'alto costo di strumenti hardware e dei pacchetti LISP allo stato dell'arte di quell'epoca
- La scarsa integrazione fra LISP e gli altri pacchetti di programmazione che rendeva difficoltosa la creazione di applicazioni embedded

L'Artificial Intelligence Section capì che l'uso di un linguaggio convenzionale, come il C, avrebbe potuto eliminare molti di quei problemi; inizialmente quindi i responsabili dei vari progetti che implicavano lo sviluppo e l'utilizzo di sistemi esperti si rivolsero a software-house già operanti in quel settore per ottenere pacchetti software scritti

in C. Nonostante le software-house iniziassero a convertire i loro prodotti in C, il costo di ciascun pacchetto software era ancora troppo elevato, senza contare il fatto che il loro utilizzo era limitato ad una ristretta varietà di computer; inoltre i tempi di progettazione erano a dir poco scoraggianti. In breve tempo divenne evidente che l'Artificial Intelligence Section avrebbe dovuto sviluppare il proprio sistema esperto basato sul linguaggio C.

Il prototipo di CLIPS fu sviluppato nella primavera del 1985 dopo soli due mesi di lavoro. Particolare attenzione fu data nel rendere il software compatibile con altri sistemi esperti e per questo motivo al sua sintassi venne resa il più vicina possibile a quella del sistema esperto ART sviluppato dall'Inference Corporation. Ad ogni modo, nonostante inizialmente fosse stato modellato sulla base di ART, CLIPS fu sviluppato interamente senza assistenza da parte dell'Inference Corporation e senza analizzare il codice sorgente di ART.

Lo scopo originale del progetto CLIPS era quello di ottenere sufficiente knowhow nella costruzione di sistemi esperti per procedere poi alla creazione di un pacchetto software capace di rimpiazzare i tool presenti in commercio sino ad allora. La versione 1.0 dimostrò la validità del progetto; dopo un'ulteriore periodo di sviluppo divenne chiaro che CLIPS sarebbe diventato un sistema esperto a basso costo, ideale per essere utilizzato a scopo didattico.

Nel 1986 fu messa a disposizione al pubblico la versione 3.0.

Ulteriori sviluppi permisero a CLIPS di passare da un semplice software prettamente didattico ad un vero e proprio tool dedicato alla creazione di sistemi esperti.

Originariamente, il primo metodo di rappresentazione in CLIPS fu un linguaggio di concatenamento delle regole basato sul Rete Algorithm. La versione 5.0 di CLIPS ha introdotto due nuovi paradigmi: il paradigma procedurale (lo stesso presente i C o FORTRAN) ed un paradigma di tipo object-oriented (presente in linguaggi come SMALLTALK). Il linguaggio di programmazione presente in CLIPS è detto CLIPS Object-Oriented Languace (COOL). [CLIPS]

3.2 Struttura interna

CLIPS ha una struttura formata da 63 moduli ciascuno dei quali ha una funzione specifica che viene utilizzata dalla shell attraverso un sistema di integrazione globale. In questa sezione viene presentata la lista di questi moduli con una loro breve descri-

zione; i moduli sono elencati iniziando da quelli che forniscono funzioni di più basso livello per arrivare a quelli che svolgono funzioni di più alto livello. Si è presa questa decisione poiché, generalmente, i moduli di più alto livello richiedono l'interazione con i moduli di livello inferiore per poter svolgere le loro funzioni specifiche.

- I primi quattro moduli (System Dependent, Memory, Symbol Manager, Router) forniscono il supporto base per moltissime operazioni di basso livello. Il modulo System Dependent implementa funzioni che dipendono in modo stretto dal sistema operativo, come ad esempio le funzioni di timing. Il modulo Memory è utilizzato per allocare e mantenere in modo efficiente le richieste di memoria. Il modulo Symbol Manager è utilizzato per evitare duplicazioni nella memorizzazione di simboli, float ed integer; impedisce inoltre che simboli, float ed integer non più utilizzati nelle applicazioni vengano memorizzati.Il modulo Router gestisce le richieste di input/output e permette a questo tipo di richieste di essere ridirette verso differenti gestori di input/output. Questa capacità di ridirezione permette la creazioe di sofisticate interfacce senza la necessità di modificare il codice del core di CLIPS.

- I successivi otto moduli (Scanner, Expression, Special Forms, Parser Utility, Evaluation, Command Line, Construct Manager, Utility) forniscolo le funzionalità base per la valutazione delle espressioni, per il supporto ai costrutti e per l'interfaccia a riga di comando di CLIPS. Il modulo Scanner legge i token da una sorgente di input. Il modulo Expression costruisce espressioni partendo dai token forniti dal modulo Scanner. Il modulo Special Forms è usato per effettuare il parsing delle funzioni che non sono conformi alla sintassi standard relativa alle funzioni che hanno a che fare con le espressioni (come ad esempio la funzione **assert**). Il modulo Parser Utility alcune funzioni utili per effettuare il parsing sia di funzioni che di costrutti. Il modulo Evaluation è in grado di valutare espressioni generate dal modulo Expression. Il modulo Command Line fornisce le funzionalità necessarie per un'interfaccia a riga di comando. E' anche in grado di determinare, prima della chiamata del modulo Expression, quando un'espressione è stata costruita, avendo come elementi di partenza solamente una serie di caratteri di input. Il modulo Construct Manager fornisce il supporto necessario per la registrazione dei costrutti così che essi possano essere riconosciuti dal parser di CLIPS. Questo modulo, in base ai diversi costrutti

che esamina, chiama le routine più aprorpiate necessarie per il caricamento ed il parsing, per il resetting e per l'inizializzazione del costrutto stesso. Il modulo Utility fornisce un gran numero di routine general purpose necessarie per visualizzare i valori, riconoscere gli errori e registrare i vari item per poterli poi utilizzare assieme al comando **watch**.

- Il modulo Fact Manager è utilizzato per mantenere la lista dei fatti (fact-list); fornisce inoltre un supporto per la creazione di valori di tipo mulifiled (attraverso funzioi come **mv-appned**). Il modulo Fact Comands implementa l'interfaccia ad alto livello per i comandi come **assert** e **facts**.

- Il modulo Deffacts fornisce le capacità necessarie per implementare il costrutto deffacts.

- Il modulo Defglobal fornisce le capacità necessarie per implementare il costrutto defglobal.

- I successivi sei moduli (Defrule Parser, Reorder, Variable Manager. Analysis, Generate, Build) sono usati per costruire le appropriate strutture dati per il costrutto defrule. Il modulo Defrule Parser è usato per effettuare il parsing della parte IF di una regola (chiamata in CLIPS Left-Hand Side - LHS) (vedi sezione 2.1), manipolando una struttura dati intermedia. Il modulo Reorder trasforma la LHS di una singola regola, contenente elementi condizionali **and** e **or**, in una struttura dati itermedia che contiene al limite non più di un singolo elemento condizionale **or** all'inizio della struttura dati stessa. Il modulo Variable Manager controlla i modelli nella LHS di una regola alla ricerca di errori semantici che coinvolgano una variabile. Inoltre questo modulo mantiene informazioni riguardanti la locazione e l'utilizzo delle variabili nei pattern di una regola. Il modulo Analysis lavora in stretto contatto con il modulo Variable Manager al fine di generare espressioni per le regole che saranno utilizzate nella rete di join and pattern. Il modulo Generate è utilizzato per generare le espressioni richieste dal modulo Analysis. Il modulo Built è utilizzato per integrare le nuove regole e le loro espressioni all'interno della rete join and pattern.

- I successivi sei moduli (Drive, Engine, Match, Retract, Rete Utility, Logical Dependencies) fromano il core del motore inferenziale di CLIPS. Il modulo Drive è utilizzato per aggiornare la rete join quando viene aggiunto un nuovo

fatto. Il modulo Engine gestisce l'agenda e l'esecuzione della parte THEN delle regole (Right-Hand Side - RHS in CLIPS) (vedi sezione 2.1). Il modulo Match determina quali pattern presenti nella rete pattern subiscono un match con un nuovo fatto inserito. Il modulo Retract è utilizzato per aggironare la rete join quando un fatto viene rimosso. Il modulo Rete Utility fornisce funzioni molto utili che vengono utilizzate dagli altri moduli per mantenere corretta la rete join. Il modulo Logical Dependencies è utilizzato per mantenere corretti i links fra la rete join ed i fatti al fine di supportare l'elemento condizionale **logical**.

- Il modulo Defrule Manager coordina l'attività di tutti i moduli usati per il mantenimento del costrutto defrule. Il modulo Defrule Deployment fornisce le funzionalità necessarie per utilizzare il costrutto defrule con i comandi **bsave**, **bload**, **constructs-to-c**. Il modulo Defrule Commands implementa l'interfaccia ad alto livello per i comandi di tipo defrule.

- Il modulo Deftemplate Command per il mantenimento dei deftemplate, fornendo un controllo di tipo e valore sugli slot dei deftemplate, fornendo l'interfaccia a alto livello per i comandi di tipo deftemplate, forndendo le utilità necessarie per utilizzare il costrutto deftemplate assieme ai comandi **bsave**, **bload** e **constructs-to-c**. Il modulo Deftemplate Function è utilizzato per effettuare il parsing dei comandi **assert**, **modify** e **duplicate** attraverso l'utilizzo del formato deftemplate. Il modulo Deftemplate Parser è usato per effettuare il parsing del costrutto deftemplate. Il modulo Deftemplate LHS è utilizzato per effettuare il parsing dei pattern di tipo deftemplate trovati nella LHS di una regola.

- Il modulo Binary Save fornisce le funzionalità necessarie al comando **bsave**, il modulo Binary Load fornisce invece le funzionalità necessarie al comando **bload**, il modulo Construct Compiler fornisce le funzionalità necessarie per il comando **constructs-to-c**.

- I successivi nove moduli (Primary Funcitons, Predicate Functions, I/O Functions, Secondary Functions, Multifield Functions, String Functions, Math Functions, Text Processing Functions, File Commands) forniscono funzioni e comandi rivolti ad una grande varietà di obbiettivi. Il modulo Primary Functions fornisce un set di comandi di ambiente e di funzioni procedurali. Il modulo

Predicate Functions fornisce un gran numero di funzioni predicative e matematiche (di tipo semplice). Il modulo I/O Functions fornisce funzioni convenienti per le operazioni di I/O. Il modulo Secondary Funcions fornise una serie di funzioni utili a fronteggiare vari tipi di necessità. Il modulo Multifield Functions fornisce un set di funzioni utili per l'utilizzo di valori multifield. Il modulo String Functions fornisce un set di funzioni necessarie per la manipolazione delle stringe. Il modulo Math Functions fornisce un set di utili funzioni matematiche che si vanno ad affiancare alle funzioni matematiche di base fornite dal moludo Predicate Funcions. Il modulo Text Processing fornise delle funzioni per la costruzione e l'accesso di un sisetma considerato gerarchico per file multipli esterni. Il modulo File Commands fornisce una serie di comandi di interfaccia che eseguono certi tipi di operazioni sui file che non sono associate con le funzioni di I/O standard fui file.ù

- Il modulo Deffuncion fornisce la capacità di definire nuove finzioni definite dall'utente direttamente in CLIPS.

- I successivi quattro moduliso sovraccaricano funzioni che possono essere definite direttamente in CLIPS: Generic Funcion Commands, Generic Function Functions, Generic Function Construct Compiler Interface, Generic Function Binary Load/Save Interface. Le funzioni di tipo generico possono fare cose diverse a seconda del tipo e del numero di argomenti che esse ricevono. Il modulo Generic Function Commands contiene la maggior parte delle routine di parsing necessarie per le funzioni di tipo generico e per i loro metodi. Il modulo Generic Function Functions determina le precedenze tra differenti metodi di una funzione di tipo generico, fornisce il generic dispatch quando viene chiamata una funzione generica; contiene inoltre altre routine di manuntenzione per le funzioni generiche ed i loro metodi. Il modulo Generic Function Construct Compiler Interface ed il modulo Generic Funcion Binary Load/Save Interface forniscono le interfacce ai comandi **bload**, **bsave** e **constructs-to-c** per le funzioni generiche.

- I prossimi dieci moduli forniscono tutte le funzionalità di COOL (vedi sezione 3.1): Class Commands, Class Functions, Instance Commands, Instance Functions, Message-Handler Commands, Message-Handler Funcions, Instance-Set Queries, Definstances, Object Construct Compiler Interface, Object Binary

Load/Save Interface. Il modulo Class Commands fornisce le routine di parsing e quelle necessarie per l'interfaccia generale necessarie al costrutto defclass. Il modulo Class Functions si prende cura di tutte le manipolazioni interne alle classi, incluso il costrutto delle liste di precedenza di classe per l'ereditarietà multipla. Il modulo Instance Commands fornise le funzioni di parsing e di interfaccia di tipo generale per le istanze delle classi definite dall'utente. Il modulo Instance Funcions si occupa di tutti i dettagli relativi alla reazione, all'accesso ed alla cancellazione di istanze. Il modulo Message-Handler Commands contiene le routine di parsing e di interfaccia generale per gli attachments procedurali alle classi. Il modulo Message-Handler Funcions implementa la spedizione dei messaggi quando un messaggio viene realmente spedito ad un oggetto e si occupa dei dettagli interni del costrutto defmessage-handler. Il modulo Instance-Set Queries fornisce le routine per un utile sistema di query che può determinare ed eseguire azioni su un set di istanze di classi definite dall'utente che soddisfano dei criteri definiti dall'utente stesso. Il modulo Definstances fornisce le capacità necessarie ad implementare il costrutto definstances. Il modulo Object Construct Compiler Interface ed il modulo Object Binary Load/Save Interface forniscono le interfacce per COOL relative ai comandi **constructs-to-c**, **bload** e **bsave**.

- Il modulo Main contiene le funzioni di startup di CLIPS.

Terminata questa descrizione di insieme di tutti i vari moduli che compongono il sistema esperto CLIPS, andiamo ora a considerare ciascuno di essi nel dettaglio.

3.3 Moduli fondamentali

3.3.1 Modulo System Dependent

Questo modulo include una serie di funzioni che contengono oggetti strettamente legati al sistema operativo o alla macchina in cui CLIPS è presente; contiene inoltre routine di inizializzazione. Durante la fase di compilazione, CLIPS per default fa in modo che le funzioni presenti in questo modulo possano essere eseguite su qualsiasi piattaforma hardware e su qualsiasi sistema operativo.

3.3.2 Modulo Memory

Durante il caricamento, la visualizzazione e l'esecuzione di programmi CLIPS vi è un costante ricorso all'allocazione di memoria. L'allocazione e la deallocazione di memoria è ottenuta attraverso un'azione indiretta a due livelli. Il primo livello permette di ottenere la separazione dalle funzioni di allocazione/deallocazione di memoria fornite dal sistema. Le funzioni **genalloc** e **genfree** forniscono il primo livello di azione indiretta. Un altro livello di questo tipo è necessario per ottenere un utilizzo efficiente della memoria. Questo secondo livello raggiunge l'efficienza traendo vantaggio dal fatto che strutture dati di dimensioni uguali fra loro sono costantemente richieste dal sistema e rilasciate al sistema ed immediatamente dopo richieste nuovamente dal sistema; è quindi chiaro che questo potrebbe portare ad un alto livello di inefficienza. Se la memoria richiesta e successivamente liberata da CLIPS fosse mantenuta dalle routine di memoria interne a CLIPS, la maggior parte del sovraccarico relativo al costante utilizzo della memoria di sistema potrebbe essere evitato.

Il modulo Memory contiene anche routine che si prendono cura di blocchi di memoria. Queste routine forniscono un altro livello di azione indiretta per la gestione della memoria. Esse richiedono grandi blocchi di memoria al fine di poter fornire memoria quando vengono effettuate richieste di allocazione di piccole porzioni. In alcune macchine queste routine possono incrementare l'efficenza della gestione della memoria se le routine di sistema sono in gradi di gestire solamente piccoli blocchi di memoria.

La gestione di CLIPS dei blocchi di memoria liberi viene ottenuta utilizzando un array di puntatori a blocchi di memoria (la **MemoryTable**). L'array degli indici si riferisce alla dimensione di memoria che viene qui immagazzinata. Richieste di memoria di dimensioni maggiori rispetto alla dimensione della **MemoryTable** sono rivolte al sistema.

3.3.3 Modulo Symbol Manager

Dati simbolici sottoforma di parole e stringhe devono essere processati in modo efficiente sia in termini di velocità, sia in termini gestione di immagazzinamento. La gestione della memorizzazione di simboli richiede che copie multiple di uno stesso simbolo vengano immagazzinate nella stessa locazione. Per raggiungere questo ri-

sultato, CLIPS utilizza una **SymbolTable** per memorizzare tutte le occorrenze dei simboli. Per esempio, il fatto *(data red green red)* potrebbe necessitare di tre entries nella **SymbolTalbe**: una per ogniuno dei simboli *data*, *red* e *green*. La **SymbolTalbe** deve anche tener traccia dei simboli che non sono più in uso e rimuoverli. Per rattiungere questo obiettivo, a ciascun simbolo viene dato un contatore per indicare il numero di referenze al simbolo stesso. Nell'esempio precedente, assumendo che non vi siano state altri ingressi nella **SymbolTable** in precedenza, il contatore relativo ai simboli *data* e *green* dovrebbe avere valore 1, mentre il contatore del simbolo *red* dovrebbe avere valore 2. Se, in qualsiasi momento, il contatore relativo ad un simbolo assume il valore 0, significa che non è più necessario mantenere il simbolo e quindi dovrebbe essere rimosso.

I simboli il cui contatore ha valore 0 vengono nominati come effimeri. Tutti i simboli effimeri vengono mantenuti in una lista; ad intervalli regolari la **EphemeralSymbolList** viene percorsa per rimuovere dalla **SymbolTable** tutti i simboli non più necessari; ciò significa che i simboli effimeri che hanno un valore del contatore maggiore di zero non verranno rimossi. Come esempio consideriamo il sequente comando:

```
CLIPS> (str-cat ‘‘red’’ ‘‘blue’’)
‘‘redblue’’
CLIPS>
```

Durante l'esecuzione di questo comando sono stati crati quattro simboli: i simboli **str-cat**, **red**, **blue** sono stati aggiunti alla **SymbolTable** quando il comando ha subito il parsing, mentre il simbolo **redblue** è stato aggiunto durante l'esecuzione del comando **str-cat**. Ciascuno di questi simboli è stato nominato effimero. Dopo l'esecuzione di questo comando, nessuno dei simboli è più necessario e possono essere tutti rimossi dalla **SymbolTable**. Consideriamo adesso il seguente esempio:

```
CLIPS> (assert (data = (str-cat ‘‘red’’ ‘‘blue’’)))
CLIPS>
```

Durante l'esecuzione di questo comando vengono creati sei simboli: *assert*, *data*, *str-cat*, *red* e *blue* che vengono inseriti all'interno della **SymbolTable** quando il comando ha subito la fase di parsing. Inoltre, il simbolo *redblue* viene aggiunto alla tabella durante la fase di valutazione della funzione str-cat. Questo comando

asserisce il fatto *(data redblue)* che contiene il simboli *data* e *redblue*. Le locazioni nella **SymbolTable** di questi due simboli vedranno il valore dei rispettivi contatori incrementare di uno come riflesso del fatto che un altro riferimento non effimero a questi due fatti è stato creato. Dopo l'esecuzione di questo comando, i simboli *assert*, *str-cat*, *red* e *blue* potrebbero essere rimossi dalla **SymbolTable**, mentre i simboli *data* e *redblue* dovrebbero rimanere.

Arrivati a questo punto si rende necessaria una precisazione: ogni simbolo che viene inserito nella **SymbolTable** viene marcato come effimero, anche se il valore del su contatore è pari a uno. Questo assicura che i simboli di carattere temporaneo creati durante il parsing dei comandi e la valutazione delle funzioni vengano rimossi in modo semplice. Un simbolo può vedere incrementato il valore del proprio contatore per una serie di svariati motivi, includendo l'utilizzo del simbolo all'interno di un costrutto (costrutti come **defrule**, **deffacts**, **defclass**) o l'utilizzo del simbolo come parte di un fatto o di un'istanza. Per quanto riguarda il decremento del valore del contatore, questo viene effettuato qualora l'oggetto corrispondente al simbolo in questione viene rimosso (per effetto della cancellazione di un costrutto o per effetto del riesame di un fatto). L'**EphemeralSymbolList** viene periodicamente verificata alla ricerca di simboli che possono essere rimossi dalla **SymbolTable**. Queste verifiche periodiche vengono effettuate in varie occasioni, per esempio dopo l'esecuzione di una regola, di una deffuncion, di una generic function, di un message-handler o di comandi ad alto livello.

A causa del fatto che i simboli possono essere creati a diversi "profondità" di valutazione (vedere sezione 3.3.9), è altresì necessario immagazzinare il valore della profondità di valutazione a cui è stato creato il simbolo in questione. I simboli effimeri non vengono cancellati a meno che il valore del loro contatore non sia nullo; inoltre, il simbolo effimero viene eliminato ad una profondità di valutazione minore rispetto a quella in cui il simbolo è stato creato.

In agginta ai simboli, anche i numeri interi ed in virgola mobile vengono immagazzinati in tabelle. I valori floating point vengono memorizzati nella **FloatTable**, mentre gli integer nella **IntegerTable**. Le operazioni su queste tabelle sono virtualmente identiche a quelle della **SymbolTable**(la più grande differenza riguarda i tipi di dato inseriti nelle tre tabelle). La **SymbolTalbe** viene utilizzata per immagazzinare i valori dei dati di tipo *symbol*, *string* ed *instance name* (per esempio, *red*, *"red"* e *[red]* hanno la stessa locazione nella **SymbolTable**). L'**IntegerTable** viene

utilizzata per immagazzinare solamente i dati di tipo *integer*, mentre la **FloatTable** immagazzina solamente i dati *floating point*.

3.3.4 Modulo Router

Il modulo Router fornisce un livello di azione indiretta che si pone fra due implementazioni di I/O di basso livello. Tutte le richieste di alto livello per l'I/O sono dirette verso nomi logici. I nomi logici sono quindi associati ad implementazioni di I/O specifiche. Per modificare l'interfaccia e passare da una di tipo linea di comando ad un'interfaccia grafica è sufficiente riassociare gli appropriati nomi logici con l'implementazione per le finestre. Le richieste di alto livello non devono essere modificate.

3.3.5 Modulo Scanner

Questo modulo effettua una scansione sulle risorse di input alla ricerca di simboli (tokens) roconoscibili da CLIPS. Lo scanner riceve gli input da nomi logici, come descritto nella sezione 3.3.4 e fornisce i dati relativi ai vari token in una struttra dati che presenta diversi campi. Uno di questi campi indica il tipo di token; per esempio, il token *783* dovrebbe essere di tipo INTEGER, il token *(* dovrebbe essere di tipo LEFT_PARENTHESIS, il token *"cat"* dovrebbe essere di tipo STRING. Un'altro campo nella struttura dati fornisce il data value dei tokens che ne possiedono uno. Ritornando all'esempio precedente, *"cat"* dovrebbe avere il data value di "cat" (che dovrebbe essere un puntatore al simbolo relativo a *"cat"* inserito nella **SymbolTable**). Degno di nota è il fatto che il simbolo *cat* dovrebbe avere lo stesso data value della stringa *"cat"*. Infine, i tokens presentano anche una rappresentazione di stampa: il token *?x*, per esempio, dovrebbe essere di tipo VARIABLE, dovrebbe avere data type "x" ed una rappresentazione di stampa pari a "*?x*".

CLIPS produce una rappresentazione formattata per ciascun comando o costrutto su cui è effettuata l'operazione di parsing. Dal momento che questo processo di formattazione è strettamente collegato con lo scanner, le routine per creare questa rappresentazione "*pretty print*" sono incluse nel modulo Scanner e chiamate direttamente dalle routine dello scanner stesso. Ciascun token che viene letto utilizzando il modulo Scanner viene inserito nel *PrettyPrintBuffer* a meno che il buffer non venga

disabilitato. Questo avviene normalmente durante l'esecuzione di una base delle conoscenze (non è infatti normalmente desiderabile formattare dati letti da un file).

3.3.6 Modulo Expression

Il formato standard utilizzato da CLIPS per le espressioni è molto simile al formato LISP. In generale, le espressioni presentano il formato:

```
(funcion-name arg1 arg2 ... argn)
```

dove ciascun argomento potrebbe essere un'espressione, un data type primitivo (simbolo, stringa, integer, float, instance name, ma non un'indirizzo esterno o un'istanza), oppure una variabile (sia locale che globale). Il nome della funzione si riferisce sia ad una funzione di sistema, sia ad una funzione definita dall'utente, una deffunction o una funzione generica. Ciascuna delle seguenti espressioni sono delle espressioni valide di CLIPS:

```
(facts)
(+ (* 3 (- ?x 3)) 6)
(str-cat "red" "blue")
```

Il modulo Expression contiene routine che effettuano il parsing delle espressioni in un formato che, in molti casi, è adatto per la valutazione da parte del modulo Evaluation (vedi sezione 3.3.9). Il modulo Expression controlla inoltre che il primo simbolo trovato nella chiamata ad una funzione sia il nome di una funzione. Il parsing di costrutti (come defrule o deffacts) viene eseguito dal modulo Construct (vedi sezione 3.3.11), mentre il parsing di alcune espressioni CLIPS che non sono conformi al formato standard vengono eseguite dal modulo Special Forms (vedi sezione 3.3.7).

La struttura dati utilizzata per immagazzinare ciascun componente di un'espressione è costituita da un campo che specifica il tipo (es. SYMBOL o INTEGER), un campo che specifica il valore (es. un puntatore ad una entry nella **SymbolTable**) ed un puntatore al prossimo argomento nella lista degli argomenti. Per esempio, la seguente espressione

```
(+ (* 3 (- 8.3 2) 11) 6.5)
```

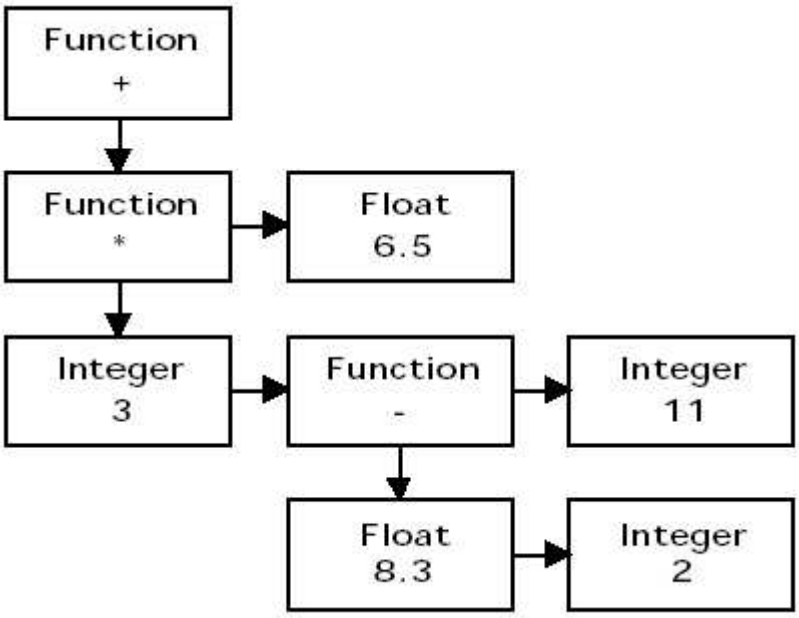

Figura 3.1: Esempio di parsing di un'espressione in CLIPS

dovrebe essere rappresentata come mostrato in figura 3.1 (le frecce discendenti rappresentano i puntatori alla lista degli argomenti, mentre le frecce rivolte verso destra rappresentano il puntatore all'argomento successivo).

3.3.7 Modulo Special Forms

Consideriamo un esempio di espressione che non è conforme al formato standard di CLIPS:

```
(assert (data 35))
```

La subespressione presente all'interno dell'assert *(data 35)* non può essere resa equivalente ad una chiamata a funzione, ma piuttosto si tratta di un insieme di dati che sevono alla funzione assert. E' quindi necessario effettuare una speciale operazione di parsing per rendere valido questo formato della funzione assert. Molte altre funzioni, come ad esempio if, while, bind, retract, trasformano le espressioni in un particolare modo od effettuano controlli addizionali sulla sintassi del formato dell'espressione. Tutte queste espressioni richiedono un parsing speciale.

Funzioni specializzate di parsing sono responsabili della costruzione di un'appropriata rappresentazione delle espressioni, allo stesso modo rispetto alla creazione di appropriate chiamate alle routine di pretty print per formare in modo corretto le espressioni per l'output.

3.3.8 Modulo Parser Utility

Questo modulo fornisce molte funzioni al fine di eseguire svariati task di parsing.

3.3.9 Modulo Evaluation

Il modulo Evaluation fornisce un set di funzioni per la valutazione delle espressioni. In aggiunta, fornisce anche funzioni per la definizione di funzioni e per accedere ai valori degli argomenti delle espressioni stesse.

Nelle versioni di CLIPS precedenti la 5.0, la raccolta di dati inutili era semplificata grazie al fatto che poteva essere effettuata al ternime del firing delle regole. I simboli e le altre strutture dati create grazie alla valutazoine di un'espressione potevano essere controllate alla fine del firing di ciascuna regla per determinare se potevano essere scartati o meno. La versione 5.0 di CLIPS ha comunque introdotto un paradigma procedurale di tipo object-oriented. E' ora possibile avere un programma CLIPS che non contiene alcuna regola, per ciò non è più sufficiente effettuare la raccolta dei dati inutili solamente dopo il firing delle regole; al contrario, ora questa attività può essere eseguita al completamento di ciascuna regola, deffuncion, funzione di tipo generico o messagle-handler che venga esegito.

A casua del firing delle regole, le chiamate a funzioni ed il passaggio di messaggi possono essere nidificati molti livelli più in basso ed è quindi necessario associare una profondità di valutazione a ciascuna struttura dati di tipo effimero che viene creata. Questa profondità di valutazione indica il livello di innestamento che deve presentarsi prima che una particolare struttura dati possa essere dichiarata inutile. Per esempio, se la funzione *foo* chiama la funzione *bar* che a sua volta chiama la funzione *yak*, allora le strutture dati create durante la valutazione delle espressioni nella funzione *foo* dovrebbero avere una profodità di valutazione pari a 1. Allo stesso modo le strutture dati relative alla funzione *bar* dovrebbero avere una profondità di valutazione di 2 e quelle della funzione *yak* dovrebbero avere una profondità di valutazione pari a 3. Le strutture dati effimere create alla profondità 3 possono essere

considerate inutili non appena si ha il ritorno alla funzione *foo* o *bar*. Similmente, le strutture dati create alla profondità di valutazione 2 potrebbero essere dichiarate inutili dopo il ritorno alla funzione *foo*, mentre le strutture dati alla profondità 1 possono essere dichiarate inutili non appena questa funzione ternima.

3.3.10 Modulo Command Line

Contiene le funzioni base per un semplice processore a riga di comando per i comandi CLIPS. Le routine di riga di comando sono orientate verso la costruzione di interfacce che usano una filosofia event-driven (guidata dagli eventi). Queste interfacce hanno finestre, menù e/o finestre per l'inserimento di comandi. In un'interfaccia event-driven, gli input da tastiera sono solamente uno dei molti possibili eventi. Se un tasto viene premuto, questo viene insertio in un buffer di input che non viene processato fin tanto che un comando complento non venga inserito. In CLIPS i comandi vengono delimitati da una serie di coppie di parentesi combacianti. In aggiunta, i comandi possono essere delle variabili o delle costanti. Durante l'inserimento del buffer di input, altri eventi, come ad esempio la selezione di un menù, possono essere processati poiché CLIPS non ha ancora iniziato a processare il comando in ingresso.

Il buffer di input di base dovrebbe essere utilizzato per accettare gli inserimenti da tastiera. Gli inserimenti da file dovrebbero essere permessi impedendo agli altri eventi di far sentire la loro presenza. In effetti, i comandi di menù non possono essere utilizzati durante il caricamento di un file. Questa operazione rappresenta un evento completo da sola, mentre l'inserimento di un singolo carattere da tastiera è un singolo evento.

Qui di seguito viene presentato il loop di comando base di CLIPS:

```
procedure CommandLoop
 print the CLIPS prompt
 do forever
  call EventFunction
  if a complete command is in the input buffer then
   perform the command
   clear the input buffer
   print the CLIPS prompt
```

```
  end if
 end do
end procedure
```

Si noti che il loop chiama la procedura **EventFuncion** ripetitivamente. Il comando viene eseguito solamente quando la funzione **CompleteCommand** indica che un comando completo sta aspettando nel buffer di input. Una tipica procedura **EventFunction** per un'interfaccia di tipo non-window potrebbe essere:

```
procedure GenericEventFunction
 get a character from the keyboard
 stuff the character into the input buffer
end procedure
```

L'unico evento possibile è acquisire un carattere che viene quindi inserito nel buffer di input. Qualora un comando viene completato, la funzione CompleteCommand ritorna un valore diverso da zero. Semplici modifiche a questa funzione di base permettono di effettuare le operazioni elementari di un'interfaccia di tipo wondow, come mostrato qui di seguito:

```
procedure WindowEventFunction
 get the next event
 if the event is a key press then
  stuff the character into the input buffer
 else if the event is a menu selection
  execute the menu selection
 else if the event is a windows operation
  execute the windows operation
 end if
end procedure
```

In questa funzione viene utilizzata una routine per ottenere il prossimo evento. A seconda dell'esatta natura dell'evento vengono poi intraprese azoini differenti. Questo tipo di setup permetterà all'utente di iniziare ad inserire un comando, navigare all'interno del databse utilizzando le opzioni di menù, e quindi finire di inserire il comando.

3.3.11 Modulo Construct Manager

In CLIPS compaiono molte definizioni di costrutti: **defrule**, **deffacts**, **deftemplate**, **defglobal**, **deffunction**, **defclass**, **definstances**, **defmessage-handler**, **defgeneric** e **defmethod**. Il modulo Construct Manager effettua molte operazioni generiche su queste strutture.

3.3.12 Modulo Utility

Il modulo Utility contiene un numero di funzioni generiche, incluse funzioni per la stampa di tipi di dato primivi, la rappresentazione di stringhe di tipi di dato primitivi, la possibilità di aggiungere in coda a stringhe altre stringhe o uncarattere, il controllo del tipo degli argomenti, la stampa di errori generici e di messaggi di informazione e per la definizione periodica dei dati inutili.

Il metoto che utilizza CLIPS per definire quali dati siano inutili merita un approfondimento. La definiizone dei dati inutili e gli elementi effimeri sono già stati presentati in modo esteso nella sezione 3.3.3. La definizione dei dati inutili in CLIPS avviene in vari modi. La prima varietà è definire come inutili i dati che possono essere immediatamente scartati quando non vengono più utilizzati Come esempio consideriamo la seguente sequenza di comandi:

```
CLIPS> (open ‘‘temp.txt’’ temp ‘‘w’’)
TRUE
CLIPS> (printout temp ‘‘Hello World’’ crlf)
CLIPS> (close temp)
TRUE
CLIPS>
```

Quando il file “temp.txt” viene aperto usanto il comando **open**, le strutture dati sono allocate associando ilnome logico temp con il nuovo file aperto. Quando viene utilizzato il comando **close** al fine di chiudere il file, le strutture dati precedentemente allocate non sono più necessarie e possono essere immediatamente restituite al pool della memoria libera. In questo caso la definizione dei dati inutili avviene nel momento stesso in cui le strutture dati di questo esempio diventano effettivamente inutili. La cancellazione di costrutti è un’altro esempio di questo tipo in quanto la memoria utilizzata da questi costrutti viene immediatamente ritornata al pool

della memeoria libera (con qualche eccezione, come la cancellazione di una regola in esecuzione).

Il secondo tipo di definizione di dati inutili avviene quando un item sembra essere inutile (ma non si può ancora determinare se ciò sia vero o meno), o quando un item è effettivamente inutile, ma esiste un riferimento temporaneo a questo item generato da un'altra struttura dati. Come esempi di questo caso, consideriamo la seguente sequenza di comandi:

```
CLIPS> (assert (colors red green))
CLIPS>
(defrule remove-fact
 ?f<-(colors ?x ?y)
 =>
 (retract ?f)
 (printout t "Colors: " ?x " " ?y crlf))
CLIPS> (run)
CLIPS>
```

Quando il fatto *(color red green)* viene ritrattato dalla regola *remove-fact*, i simboli *red* e *green* diventano inutili in quanto non esistono più riferimenti permanenti reattivi ad altre strutture dati. Ad ogni modo, questi valori sono ancora necessari al comando *priotout* che segue il comando *rectract*, così i valori non possono ancora essere dichiarati inutili. Non appena l'esecuzione della regola è completa, i valori possono essere dichiarati inutili in tutta sicurezza.

La definizione di dati inutili può avvenire al completamento di ciascuna regola, deffuncion, funzione di tipo generico o message-handler che è in esecuzione; comunque non deve necessariamente avvenire ogni volta che una di queste operazioni termina la propria esecuzione. CLIPS utilizza un metodo euristico per determinare se deve iniziare la ricerca di dati inutili o meno. Innanzitutto, sia la dimensione che il numero dei dati inutili deve superare un valore specifico. In secondo luogo, questa operazione non deve essere ripetuta a questo livello di profondità finché non venga creato un numero maggiore di dati inutili. Questo evita di operare su elementi che non possono ancora essere dichiarati inutili.

3.3.13 Modulo Fact Manager

Fornisce le funzionalità necessarie per mantenere, aggiornare e navigare attraverso i fatti. Esso fornisce inoltre un'implementazione di alto livello per i comandi *assert* e *retract*. Le funzioni per visualizzare e navigare attraverso i fatti sono fornite allo stesso modo. L'altra grande funzionalità fornita da questo modulo è una tabella hash contenente i fatti. A differenza di OPS5, il paradigma inferenziale di CLIPS non permette due occorrenze dello stesso fatto nella fact-list. Una tabella hash di fatti fornisce un metodo conveniente per determinare se un fatto è già presente nella fact-list.

3.3.14 Modulo Fact Commands

Il modulo Fact Commands fornisce un numero di comandi per la manipolazoine e l'esame di fatti. I comandi forniti sono: **assert**, **retract**, **save-facts**, **load-facts**, **fact-index**, **dependencies**, **dependents**, **set-fact-duplication** e **get-fact-duplication**.

3.3.15 Modulo Deffacts Manager

Questo modulo controlla tutti gli aspetti del costrutto deffacs incluso il parsing, l'esecuzione e la rimozione.

3.3.16 Modulo Defglobal Manager

Il modulo Defglobal Maganer si prende cura di tutti gli aspetti relativi al costrutto defglobal, incluso il parsing, l'esecuzione e la rimozione.

3.3.17 Modulo Defrule Parser

Il modulo Defrule Parser coordinta il parsing della LHS delle regole (allo stesso modo nel quale fornisce le funzioni per il parsing della RHS di una regola). Gli elementi condizionali della LHS sono rappresentati internamente utilizzando un formato mostrato in figura 3.2.

Se l'elemento condizionale è un **test** CE (Conditional Element), l'informazione CE sarà un'espressione immagazzinata ustando il formato standard di un'espressione. L'informazione CE per un elemento condizionale connesso (un **and** CE oppure

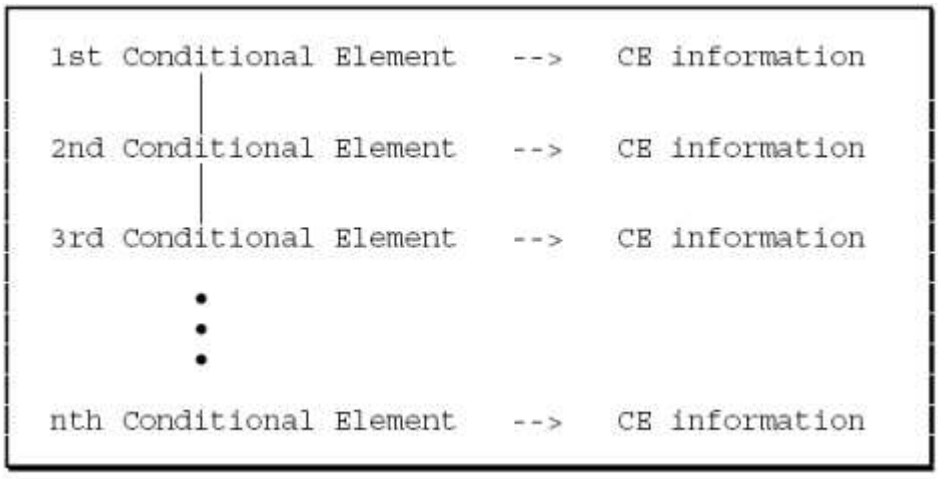

Figura 3.2: Formato di rappresentazione degli elementi condizionali della LHS nel modulo Defrule Parser

un **or** CE od un **logical** CE) segue il formato mostrato in precedenza. L'informazione per un **pattern** CE o un **not** CE normalmente rappresenta i campi del pattern. Come esempio, gli elementi condizionali della seguente regola

```
(defrule example
 (pattern 1)
 (or pattern 2a)
    (pattern 2b))
 (not (pattern 3))
 (pattern 4)
=>)
```

dovrebbero essere immagazzinati come mostrato in figura 3.3.

L'informazione CE per i **pattern** CE e per i **not** CE è memorizzata ustando il formato mostrato in figura 3.4.

L'informazione per ciascun campo è memorizzata nel formato indicato in figura 3.5.

Il primo first-binding di una variabile è memorizzato nella struttura (se esiste). Un first-binding di una variabile per un campo in un pattern è una variabile essa stessa oppure una variabile seguita da un costrutto connettivo &. La variabile non

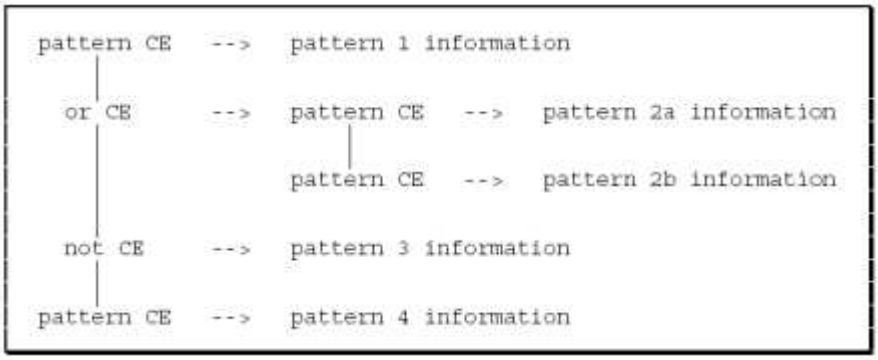

Figura 3.3: Esempio di rappresentazione degli elementi condizionali della LHS di una regola

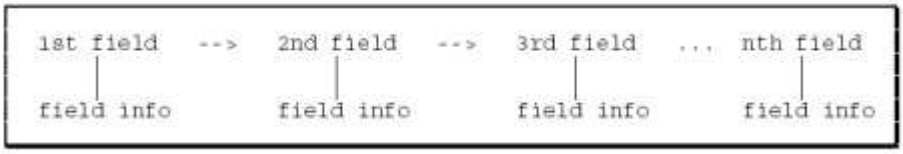

Figura 3.4: Formato di memorizzazione dei pattern CE e not CE

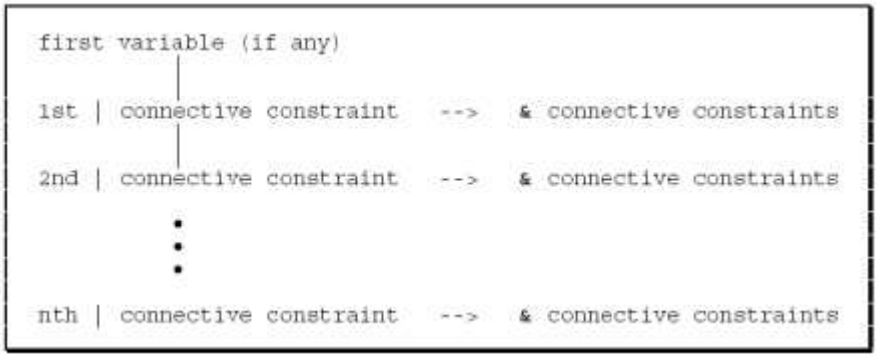

Figura 3.5: Formato di memorizzazione di un campo all'interno del modulo Defrule Parser

può essere negata. La prima volta che si incontra la variabile ?x in un campo di un pattern si dovrebbe osservare uno di questi casi:

```
?x
?x&blue|green
~?x
?x|?y
red&?x
```

La struttura che deve contenere la variabile interessata dal firs-binding viene anche utilizzata per indicare se il campo dovrebbe effettuare un match con un campo singolo o multivalore. I campi che non hanno una variabile assegnata vengono considerati per i soli match con campi singoli.

Il link inferiore successivo connesso alla struttura della variabile assegnata contiene informazioni riguardo la lista di strutture connettive "|" all'interno del campo. Si può accedere a ciascun costrutto connettivo "|" di un campo attraverso il link inferiore della struttura. Le strutture alla sua destra rappresentano altri costrutti associati con il simbolo connettivo "|" attraverso l'utilizzo di strutture connettive "&". I costrutti, singolarmente, possono essere di tipo letterale, possono essere di tipo predicativo oppure essere utilizzati per ritornare dei valori oppure ancora possono essere dei costrutti relativi a variabili. Ciascuno di essi può essere negato utilizzando il costrutto connettivo "~". Quando vengono raggruppati, i costrutti connettivi "|" presenti all'interno di un campo di un costrutto ricevono una priorità minore rispetto ai costrutti connettivi "&". Per esempio, il seguente campo:

```
?x&:numberp(?x)&=(+ ?y 3)|:wordp(?x)&~red&~green
```

dovrebbe essere rappresentato come mostrato in figura 3.6.

3.3.18 Modulo Reorder

La topologia Rete di base (vedi sezione 2.1.1) permette solamete patterns di elementi condizionali isolati o che vengano modificati da elementi condizionali **not**. Inoltre la LHS è chiuso da un elemento condizionale **and**. Utilizzando la topologia Rete di base non è permessa la combinazione fra elementi condizionali **and** ed elementi condizionali **or**. CLIPS permette che questi due tipi di elementi condizionali vengano combianti fra loro generando regole multiple che sono conformi alla topologia Rete

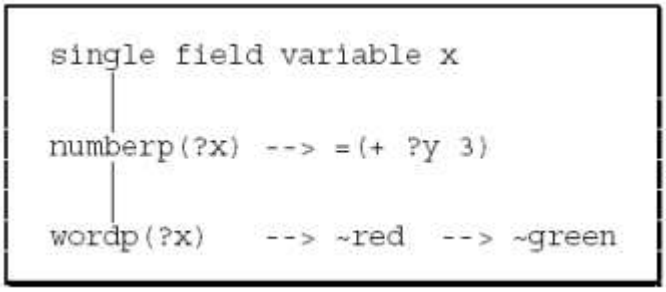

Figura 3.6: Esempio di rappresentazione di un campo di un costrutto da parte del modulo Defrule Parser

di base. Il modulo Reorder riordina una singola LHS, che può essere conforme o meno alla topologia Rete di base, in una o più LHS che sono sicuramente conformi a questa topologia. Le LHS riordinate hanno un singolo elemento condizionale **or** di alto livello all'interno del pattern (con ciascun argomento dell'elemento condizionale **or** rappresentante una regola separata che deve essere generata) con elementi condizionali **and** multipli contenenti uno o più elementi condizionali di pattern o elementi condizionali **not**. Allo scopo di effettuare il riordino, l'elemento condizonale **logical** si comporta esattamente come l'elemento condizionale **and**.

Il riordinamento coinvolge due ulteriori steps: trasformazione e riduzione. La trasformazione implica il cambiamento di un elemento condizionale da una forma ad un'altra equivalente. La trasformazione viene effettuata quando riordinare i patterns inplica rimpiazzare elementi condizionali **and**/**or** con elementi condizionali **and**/**or** equivalenti. Per esempio,

```
(and (or (a) (b))
     (or (c) (d)))
```

può essere rimpiazzato con

```
(or (and (a)
         (or (c) (d)))
    (and (b)
         (or (c) (d))))
```

Questa trasformazione sfrutta il fatto che gli elementi condizionali contenuti all'interno di *(or (a) (b))* possono essere estratti e combinati individualmente con un

elemento condizionale **and** con copie di *(or (c) (d))*. Il risultato può quindi essere riunito utilizzando un elemento condizionale *or*. Consideriamo ora il caso generale relativo a questa trasformazione:

```
(and(<CE-a-1> ...
     (or <CE-o-1> ... <CE-o-n>) ...
     <CE-a-n>)
```

che può essere sostituito con:

```
(or (and <CE-a-1> --- <CE-o-1> ... (pattern an)) ...
    (and <CE-a-1> --- <CE-o-n> ... (pattern an)))
```

La riduzione coinvolge invece la semplificazione degli elementi condizionali. La riduzione, utilizzata quando gli elementi condizionali vengono riordinati, coinvolge la rimozione di informazioni ridondandti. Per esempio, un Ce come ad esempio *(and (and (<CE-1> <CE-2>)* può essere semplificato in modo da ottenere *(and <CE-1> <CE-2>)*. Questo tipo di riuzione è detto riduzione di adiacenze. Consideriamo l'esempio ulteriore costituito da:

```
(or (and (and (a) (b)) (and (c) (d))))
```

che può essere convertito in:

```
(or (and (a) (b)) (and (c) (d)))
```

Bisogna sottolineare il fatto che per questo tipo di semplificazione gli elementi condizionali **and**/**or** devono essere adiacenti.

3.3.19 Modulo Variable Manager

Il modulo Variable Manager fornisce un set di funzioni di accesso che sono usate per reperire i risulati dell'analisi della LHS di una regola. Alcune delle funzini fornite possono essere utilizzate per determinare la locazione di una variabile nella LHS di una regola e per ottenere l'espressione generata dalle reti pattern e join. Queste funzioni di accesso sono utilizzate dal modulo Built (vedi sezione 3.3.22) quando aggiunge una regola alla rete delle regole.

3.3.20 Modulo Analysis

Il modulo Analysis crea le appropriate chiamate a funzioni al fine di inserirle nelle reti join e pattern. Inoltre utilizza sia il modulo Variable (vedi sezione 3.3.19) che il modulo Build (vedi sezione 3.3.22) al fine di creare espressioni per poterle inserire nelle reti. Quando la rappresentazione della LHS di una regola viene passata alle funzioni di analisi delle regole, si succedono molti step per la generazione di un'espressione.

Il modulo Analysis determina la locazioni delle variabili all'interno dei pattern e se sono presenti errori che riguardano l'utilizzo delle variabili stesse. Analizza il set di pattern della LHS al fine di determinare dove le variabili iniziano ad essere assegnate. Tiene traccia dei pattern assegnati a variabili di indirizzamento di fatti e determina la presenza di errori di utilizzo di queste variabili.

Ciascun pattern vede memorizzate le seguenti informazioni: di che pattern si tratta (primo, secondo, terzo...)? Il pattern viene assegnato ad una variabile di indirizzamento di fatti? Se la risposta è affermativa, qual'è il nome della variabile? Il pattern viene logicamente negato? Quali variabili sono assegnate in questo pattern?

Le variabili assegnate sono variabili che possono essere sole all'interno di un campo oppure possono essere il primo elemento di un campo, seguite immediatamente da un costrutto connettivo &. Di queste variabili vengono memorizzate le seguenti informazioni: nome della variabile, il numero del pattern e del campo nei quali la variabile è stata trovata.

L'errore tipico rilevato dal modulo Analysis è il riferimento ad una variabile che deve essere ancora assegnata. Negli esempi seguenti si ha sempre un riferimento incorretto alla variabile *?x*:

```
(defrule error1
 (fact ~?x)
 =>)
(defrule error2
 (not (fact ?x))
 (data ?x)
 =>)
(defrule error3
 (data ?y)
```

```
 (test (> ?x ?y))
 =>)
(defrule error4
 (not fact ?x))
 (test (>?x 4))
 =>)
(defrule error5
 (data ?x | all)
 =>)
```

Le regole *error1* ed *error3* fanno semplicemente riferimento alla variabile *x* prima che essa venga assegnata. Le regole *error2* ed *error4* dimostrano che lo scopo del primo assegnamento di una variabile all'interno di un **not CE** è strettamente limitato all'interno del **not CE** stesso. La regola *error4* può essere corretta inserendo il *test* all'interno del **not CE** usando *&:(> ?x 4)*. Anche la regola *error5* fa riferimento alla variabile ?x quando essa non è ancora assegnata. La variabile ?x è la prima variabile nel campo; comunque essa è connessa con un costrutto connettivo | e per questo non è considerata essere un assegnamento per la variabile.

E' da notare il fatto che il costrutto deftemplate genera patterns LHS "normali" dai template LHS pattern usato in una regola. A causa del fatto che i campo possono essere elencati in qualsiasi ordine in un template pattern, è possibile per un template pattern convertito accedere ad una variabile prima che la variabile sia definita. Per esempio, consideriamo il seguente deftemplate:

```
(deftemplate temp
 (field x)
 (field y))
```

Le seguenti due regole utilizzano propriamente i template pattern:

```
(defrule example-1
 (temp (x ?x) (y ?y&~?x))
 =>)
(defrule example-2
 (temp (y ?y) (x ?x&~?y)
 =>)
```

Bisogna evidenziare che nei template pattern di entrambe le regole le variabili sono definite prima che vengano utilizzate. Comunque quando viene effettuata la conversione dai LHS template pattern ai LHS pattern normali, le regole assumono la seguente forma:

```
(defrule example-1
 (temp ?x ?y&~?x)
 =>)
(defrule example-2
 (temp ?x&~?y ?y)
 =>)
```

La regola *example*-1 non inoltra riferimenti a variabili; ad ogni modo la regola *example-2* fa riferimento alla variabile *?y* prima che essa sia definita. A causa del fatto che i campi in un template pattern possono essere specificati in qualsiasi ordine ed anche del fatto che i campi specificati possono essere riarrangiati nella generazione del vero LHS pattern da utilizzare, i riferimenti a variabili inoltrati in un template pattern sono permessi fin tanto che le variabili sono contenute da qualche parte all'interno del pattern nel quale si trova il primo riferimento ad esse.

Dopo che le variabili all'interno dei pattern sono state identificate, il modulo Generate (vedi sezione 3.3.21) può essere utilizzato per generare le espressioni per le reti patterns e join. Molti fattori devono essere considerati quando si genera delle espressioni la valutazione all'interno delle reti. Consideriamo alcuni esempi per chiarire quest'ultima afermazione.

```
(defrule example1
 (foo ?x)
 (not (bar ?x))
 (test (> ?x 4))
 =>)
```

La regola *example1* dimostra che due separate espressioni possono essere necessari per i join con un **not CE**. La prima di queste espressioni è rappresentata dallo stesso pattern. Questa espressione effettua controlli per vedere se *?x* nel fatto *bar* è la stessa *?x* presente nel fatto *foo*. L'espressione *(> ?x 4)* fa riferimento a *?x* nel fatto *foo*, ma non dovrebbe essere associata con altra espressione. Questo è necessario

nel caso in cui non esistano fatti *bar*. L'espressione che confronta *?x* in *foo* e *?x* in *bar* non deve essere eseguita in questo caso. Se l'espressione *(> ?x 4)* fosse stata associata all'altra espressione. l'esistenza di un qualsiasi fatto *foo* avrebbe causato l'attivazione della regola, anche se non sono presenti fatti *bar*. La regola dovre essere attivata solamente con la presenza di fatti *foo* con *?x* maggiore di 4.

```
(defrule example2
 (foo ?x)
 (bar ?x ?x)
 =>)
```

La regola *example2* ha due espressioni che devono essere eseguite dal secondo pattern. La prima espressione assicura che *?x* nei fatti *bar* sia la stessa *?x* nei fatti *foo*. La seconda espressione assicura che *?x* nel secondo campo dei fatti bar sia la stessa *?x* presente nel primo campo. L'espressione di comparazione attraverso i pattern deve essere fatta nella join network. L'espressione di comparazione delle due *?x* nei fatti *bar* può essere eseguita nella rete pattern.

```
(defrule example3
 (foo ?x)
 (bar ?x | all)
 =>)
```

La regola example3 è un esempio di un'espressione che deve essere spostata dalla rete pattern alla rete join. A causa del fatto che il secondo campo nei fatto *bar* deve essere comparato con il valore del primo assegnamento in un altro pattern, l'espressione per questo campo deve essere spostata all'interno della rete join.

```
(defrule example4
 (foo ?x&:(numberp ?x))
 =>)
```

La regola *example4* è un'altro esempio di un'espressione che può essere valutata nella rete pattern poiché tutti gli argomenti di *numberp* possono essere raggiunti nel pattern stesso.

```
(defrule example5
```

```
(bar ?y)
(foo ?x&:(> ?x ?y))
=>)
```

La regola *example5* mostra un costrutto predicato che deve essere valutato nella rete join a causa del riferimento a *?y* assegnato al di fuori del pattern.

Il nodulo Analysis genera un'espressione per la rete pattern per ciascun campo in un pattern ed un'espressione per la rete rete join per ciascun pattern. Anche i *not CE* possono avere un'espressione addizionale per la rete join. Il modulo Analysis determina quali espressioni sono eseguite nella rete pattern e quali sono eseguite nella rete join. Quando possibile, le espressioni dovrebbero essere valutate nella rete pattern.

Se un particolare campo non ha costrutti connettivi |, si applicano poche restrizioni all'espressione che deve essere eseguita nella rete pattern.Tutti i test rivolti alle costanti possono essere valutati nella rete pattern. I costrutti predicati ed i costrutti relativi ai valori ritornati possono essere valutati nella rete pattern fin tanto che non vengono fatti riferimenti a variabili presenti in altri pattern. Le espressioni che confrontano due riferimenti alla stessa variabile possono essere valutati nella rete pattern se entrambi i riferimenti sono stati rilevati nello stesso pattern ed uno dei due riferimenti non è un'assegnamento alla variabile. Tutte le altre espressioni che referenziano variabili al di fuori del pattern devono essere fatte nella rete join. Bisogna sottolineare che i **test CE** sono sempre eseguiti nella rete join mentre i costrutti predicativi ed i costrutti relativi ai valori ritornati possono essere eseguiti sia nella rete pattern che nella rete join; ciò dipende dai riferimenti alle variabili.

Se un campo ha un costrutto connettivo | al suo interno ed i riferimenti sono rivolti a variabili assegnate in un'altro pattern, tutte le espressioni per quel campo devono essere eseguite nella rete join.. La regola *exemple3* è un esempio di questo ultimo tipo di espressioni.

3.3.21 Modulo Generate

Il modulo Generate trasforma la sintassi delle primitive di un pattern in un'espressione che sarà inserita nelle reti pattern e join. i construtti connettivi **&** e | sono sostituit rispettivamente l'equivalente chiamata alla funzione **and** o alla funzione **or**.

L'accesso a variabili specifiche dalla rete join o dalla RHS usa la funzione **getvar**.

```
(getvar <pattern-m> <field-n>)
```

L'accesso a variabili specifiche dalla rete pattern utilizza la funzione **getfield** che richiede solamente di specificare un campo in quanto il pattern specifico è definito dal fatto corrente.

```
(getfield <field-m>)
```

La comparazione fra variabili nella rete join utilizza **eqvar** e **neqvar**, rispettivamente, a seconda del fatto che un set di variabili sia eguale o diseguale. Il pattern associato al primo campo nella comparazione è assunto essere il pattern in ingresso dalla RHS del join nel quale l'espressione è localizzata. La profondità di campo della struttura join è utilizzata per determinare questo pattern index.

```
(eqvar field-n pattern-x field-y)
(neqvar field-n pattern-x field-y)
```

Le costanti sono valutate nella rete pattern utilizzando le funzioni **constantn** e **notconstant**.

```
(constant <value>)
(notconstant <value>)
```

Le costanti sono valutate nella rete join utilizzando le seguenti funzioni (da notare il fatto che le chiamate a **eq**, **neq** e **getvar** potrebbero essere rimossi in favore di una chiamata ad una funzione single-level):

```
(eq (getvar <pattern> <field>) <value>)
(neq (getvar <pattern> <field>) <value>)
```

La primitiva di pattern

```
=(expression)
```

è sostituita con

```
(eq (getvar <pattern> <field>) (expression))
```

nella rete join, e con

```
(eq (gerfield <pattern>) (expression))
```

nella rete pattern. Per una comparazione di diseguaglianza *(~=)* posso essere utilizzate neq e neqfield. L'espressione primitiva

```
:(expression)
```

è sostituita con

```
(expression)
```

La rete join utilizza le chiamate a **getvar** per risolvere le refernziazioni, mentre la rete pattern utilizza per lo stesso scopo le chiamate a **getfield**.

3.3.22 Modulo Build

Le informazioni e le espressioni generate durante la fase di analisi della compilazione di una regola devono essere integrate all'interno delle reti pattern e join. Questa ingegrazione trae vantaggio dal potenziale dovuto alla condivisione delle espressioni comuni attraverso i vari pattern e join, dove possibile. Il modulo Build utilizza le informazioni create dal modulo Analysis (vedi sezione 3.3.20) e gli accessi garantiti dal modulo Variable Manager (vedi sezione 3.3.19) al fine di aggiungere una regola nella rule network (coustituita dalle rete pattern e join).

La rete pattern è concettualmente rappresentata da una struttura ad albero. Il nodo di root rappresenta il punto di partenza di un pattern match prima che un qualsiasi elemento di un qualsiasi fatto o della rete pattern venga "consumanto". Il set di nodi che seguono quello di root rappresentano tutte le pattern expression trovate come primo campo in un pattern. I figli di questi nodi rappresentano tutti i campi trovati come secondi in un pattern. Ciascun livello del pattern tree rappresenta il set di tutti i campi di una particolare espressione in tutti i pattern. Percorrendo l'albero dall'alto verso il basso vengono consumati i campi dei fatti, sotto forma di espressioni che vengono valutate. La struttura della pattern network consente ai pattern di condividere identiche sequenze di campi che iniziano di fronte al pattern. I due pattern

```
(data red ?)
(data green ?)
```

dovrebbero condividere un nodo di pattern comune a causa del loro primo campo, *data*. Se il pattern

```
(data green blue ?)
```

viene ora aggiunto, anch'esso dovrebbe condividere il nodo comune *data* con i due precedenti campi. Inoltre dovrebbe condividere anche il nodo *green* con il secondo pattern.

Affinché si possa varificare una condivisione di campi devono essere rispettate due condizioni: innanzitutto tutti i campi precedenti nel pattern devono essere condivisi nella pattern network. In secondo luogo, le espressioni generate partendo dai campi del pattern devono essere identici ad un'espressione già presente nel livello corrente nella rete pattern. Bisogna notare che generalmente le variabili non creano espressioni che sono testate nella pattern network, a meno che non si riferiscano a variabili assegnate precedentemente nello stesso pattern.

Per quanto riguarda la join network, vi sono tre condizioni da soddisfare affinché si possa verificare la condivisione di un join di una regola: in primo luogo tutti i join precedenti devono essere condivisi. In secondo luogo le espressioni generate per il join devono essere identiche ad un'espressione già presente nella join network al livello corrente. L'ultima condizione prevede che il join da condividere deve essere inserito dalla stessa location nella pattern network. Consideriamo i seguenti due esempi:

```
(defrule example1
 (data red ?x)
 (data green ?x)
 =>)
(defrule example2
 (data red ?x)
 (data blue ?x)
 =>)
```

Tutti e quattro i pattern condividono il nodo comune *data*. Inoltre, il primo pattern in entrambe le regole può condividere tutti i pattern node. Anche il join per questi primi pattern in entrambe le regole può essere condiviso, contrariamente al secondo join che non può essere condiviso. Infatti, se da un lato l'espressione del join è identica per entramber le regole (corrisponde a *(eqvar 2 1 2)*), dall'altro i join devono essere inseriti da differenti pattern e quindi non possono essere condivisi.

La seguente regola

```
(defrule example3
 (data red ?x)
 (data blue ?y)
 (info ?x)
 =>)
```

dovrebbe essere in grado di condividere nodi nella rete pattern e nella rete join. I suoi primi due pattern sono già presenti all'interno della pattern network. Il terzo pattern richiede l'aggiunta di due nuovi nodi nella pattern network. I primi due join di questa regola potrebbero essere condivisi con i join usati per la regola *example2*. Un terzo joinper la regola *example3* dovrebbe essere aggiunto per l'ultimo pattern. Bisogna sottolineare il fatto che l'utilizzo di nomi diversi per le variabili non influenza la capacità di condivisione nelle due reti. Fin tanto che le espressioni generate sono identiche si verificherà la condivisione. I nomi delle variabili sevono solamente come riferimenti posizionali.

Le informazioni riguardanti la condivisione nella rete join possono essere visualizzate quando una regola viene caricata, a patto che venga settato il flag di compilazione del watch. Nuovi inserimenti nella join network vengono segnalate grazie a $+j$, mentre il riutilizzo di nodi esistenti è indicato con $=j$.

La topologia di join di CLIPS differisce leggermente dalla topologia standard Rete usata da OPS5. Innanzitutto, ciascun pattern corrisponde al proprio join. Nella topologia standard, i primi due pattern formerebbero un join two-input. Se essite un solo pattern, questo formerà un singolo join one-input. In CLIPS il primo pattern crea sempre un join one-input. Questo semplifica considerevolmente l'algoritmo utilizzato dal momento che un pattern viene sempre inserito nel join dalla RHS. Dando questa semplificazione, la memoria beta di un join non può mai essere associata con un pattern contenuto in un **not CE**.

La topologia standard Rete effettua inoltre un test nella pattern network sulla lunghezza di un fatto. L'utilizzo di variabili multicampo in CLIPS elimina molti dei vantaggi derivanti da questo test. Ciascun livello della rete pattern corrisponde direttamente ai campi indicizzati in un pattern.

3.3.23 Modulo Drive

Il modulo Drive contiene le maggiori funzionalità per l'aggiornamento della rete join quando un fatto viene inserito all'interno del database delle conoscenze. Questo aggiornamento verrà anche visto come la conduzione di match parziali attraverso la join network. Quando un fatto ha generato un match con un pattern nella pattern network, viene creato un match parziale consistente in quel singolo fatto. Questo match parziale viene quindi spedito ad ogni singolo join della join network che risulti connesso con quel pattern. Il match parziale entra attraverso la RHS del join. A seconda del tipo di join il match parziale in ingresso sarà comparato con la memoria beta del join. Nuovi match parziali possono essere creati da questo confronto e spediti ai join discendenti dal join corrente. Nuovi match parziali potrebbero entrare dalla LHS dei join discendenti. Bisogna notare che tutti i pattern degli elementi condizionali entrano nei join dalla RHS, ma tutti i join entrano in altri join dalla LHS. L'algoritmo per condurre i match parziali attraverso la join network è descritto qui di seguito.

La funzione **Drive** si occupa di aggiornamenti di alto livello della rete join quando un fatto viene asserito. Sei il join che viene aggiornato è un join terminatore (cioè l'ultimo join di una regola che connete il join con le azioni di una regola), è stata eseguita l'attivazione di una regola. Un'attivazione viene insertia nell'agenda ed il livello corrente di ricorsione della rete join è terminato. Se il join era entrato dalla LHS, il match parziale viene memorizzato nella memoria beta del join. I match parziali entrati dalla RHS sono già stati immagazzinati nella memoria alfa all'interno della rete pattern.

Se il join aggiornato è un join single-entry (per esempio il join associato con il primo elemento condizionale in una regola), allora viene utilizzato l'algoritmo single-entry join al fine di aggiornare la rete join. Innanzitutti viene determinato se l'espressione primaria del join ha valore TRUE o FALSE. se il valore è FALSE l'aggiornamento è terminato.

Se l'LHS entry è un elemento condizionale **pattern**, allora viene effettuata una copia del match parziale alfa e viene spedita a tutti i join figli del join corrente utilizzando l'algoritmo **Drive**.

Se l'LHS entry proviene da un elemento condizionale **not** ed il valore del contatore dei fatti che rispettano il CE è maggiore di zero, allora il contatore associato al join

viene incrementato di uno.

Per i double-entry join viene utilizzato un loop al fine di comparare ciascun set di match parziali nell'opposta memoria (la memoria beta per RHS e la memoria alfa per LHS) con il match parziale in ingresso. Se il join è entrato dalla RHs, ciascun match parziale nella memoria beta viene comparato con il match parziale entrante. Se il join è entrato dalla LHS, viene comparato ciascun join nella memoria alfa (memorizzata nella rete pattern) con il match parziale entrante. Per ciascuna coppia di match parziali che viene comarata, viene verificata l'espressine del join primario per stabilire se sia TRUE oppure FALSE. Se non esiste nessuna espressione, la valutazione viene considerata TRUE. Se l'espressione di join risulta TRUE, uno dei tre algoritmi viene eseguito. Questi tre algoritmi corrispondo ai seguenti casi: positive RHS entry e positive LHS entry, positive RHS entry e negative LHS entry (significa che l'elemento condizionale associato con questo join è un **not**) con il match parziale entrante dalla LHS, positive RHS entry e negative LHS entry con il match parziale entrante dalla RHS. Gli algoritmi per ciascuno dei tre casi sono descritti qui di seguito.

L'algoritmo double-entry join per i join con una positive RHS entry ed una positive LHS entry lavora come segue. I match parziali beta e alfa sono fusi per creare un nuovo match parziale con il match parziale alfa legato alla fine del match parziale beta. Questo nouvo match parziale viene quindi spedito a tutti i join figli del join corrente usando l'algoritmo **Drive**.

Descriviamo adesso l'algoritmo double-entry join per i join associati con un elemento condizionale **not** nel quale un match parziale entra dalla LHS. Il valore del contatore associato con il match parziale beta viene incrementato di uno. Questo contatore è originariamente settato a zero quando il match parziale join entra nel join. Se, al termine di tutte le comparazioni di memoria, il valore di questo contatore è ancora nullo, viene creato un match parziale attraverso la fusione del match parziale beta con un ID di uno pseudo-fatto.

Consideriamo adesso l'algoritmo double-entry join per i join associati con un elemento condizionale **not** nel quale un match parziale entra dalla RHS. Se il valore del contatore associato la match parziale della memoria beta è maggiore di zero viene incrementato di uno. Se il valore del contatore è minore di zero, viene settato ad uno il contatore della memoria beta e vengono rimossi tutti i match parziali contenenti l'ID dello pseudo-fatto associato precedentemente al match parziale beta da tutti i

join discendenti dal join corrente.

Subito dopo il completamento della comparazione fra la memoria alfa e beta viene effettuato un test finale. Se la RHS join entry è associata con un elemento condizionale **not**, se il join è entrato dalla LHS ed il numero di match parzial alfa che soddisfano l'espressione del join primario è nullo, allora potrebbe essere necessario creare un nuovo match parziale. Viene quindi valutata l'espressione del join secondario. Se la valutazione è TRUE viene creato un match parziale consistente in un numero di identificazione di uno pseudo-fatto e dal match parziale beta. L'ID dello pseudo-fatto è negativo. Il valore del contatore associato al match parziale beta viene settato al valore dell'ID dello pseudo-fatto. Il valore del contatore rappresenta il numero di match parziali alfa che soddisfamo l'espressione del join per un particolare match parziale beta. Un valore negativo indica che non vi sono match parziali alfa che soddisfano l'espressione di join. Il nuovo match parziale creato dall'ID dello pseudo-fatto e dal match parziale beta viene spedito a tutti i join discendenti dal join corrente usando l'algoritmo **Drive**.

Consideriamo come esempio la seguente regola:

```
(defrule match ‘“’
 (point ?x ?)
 (point ? ?x)
 =>)
```

Il diagramma in figura 3.7 mostra la join network per questa regola.

Le due scatole rappresentano i join. La scatola in alto è il join associato al first pattern CE *(point ?x ?)*. La scatola in basso è il join associato con il second pattern CE *(point ? ?x)*. Le linee che hanno come terminatore un punto nero a sinistra di ciasscun join rappresentano il contenuto della memoria beta dei join. Come mostrato in questa figura, la memoria beta di entrambi i join è vuota. Il cerchio rappresenta il nodo pattern rappresentante il completamento del pattern *(point ? ?)*. Questo pattern è utilizzato sia per *(point ?x ?)* che per *(point ? ?x)* poiché le variabili in questo caso non possono essere controllate nella rete pattern. La memoria alfa è rappresentata dalla linea che ha come terminatore un punto nero connessa a destra del noto pattern. Anche la memoria alfa è vuota. Vale la pena evidenziare il fatto che le espressioni associate a ciascun join non vengomo mostrate. Il top join non ha espressioni e permetterà a qualsiasi match parziale di filtrare verso il prossimo join

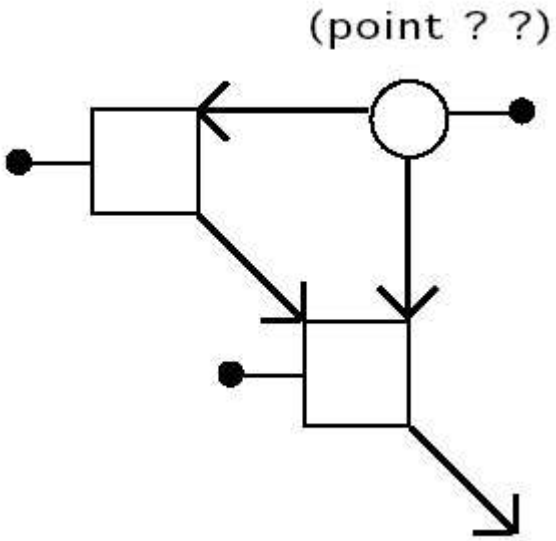

Figura 3.7: Esempio di rete join per una regola di CLIPS. Schema 1

posto in basso. Il join posto in fondo alla rete deve verificare che il valore del secondo campo del fatto assegnato al primo pattern sia eguale al valore del terzo campo del fatto assegnato dal secondo pattern. Questo diagramma presuppone che la regola di match sia la sola regola nella rule network (quindi non vi sono condivisoni). Il join terminatore della regona non è mostrato nel diagramma. Quest'ultimo join immagazzina i match parziali che soddisfa tutti i CE della regola. Viene anche creato un link tra questi match parziali e le loro corrispondenti attivazioni nell'agenda.

Supponiamo ora che venga inserito all'interno della lista dei fatti questo fatto:

```
f-1  (point 3 4)
```

Questo fatto dovrebbe soddisfare il pattern *(point ? ?)* e dovrebbe essere inserito nella memoria alfa del nodo pattern. Il nodo pattern deve passare questo match parziale ai due join a cui è connesso. Per primo il match parziale viene spedito al top join, come mostrato in figura 3.8.

Dal momento che il top join non ha espressioni, il match parziale scende al prossimo join (vedi figura 3.9).

Giunti a questo punto si verifica una comparazione tra il match parziale nella memoria beta e tutti i match parziali presenti nella memoria alfa che sono associati

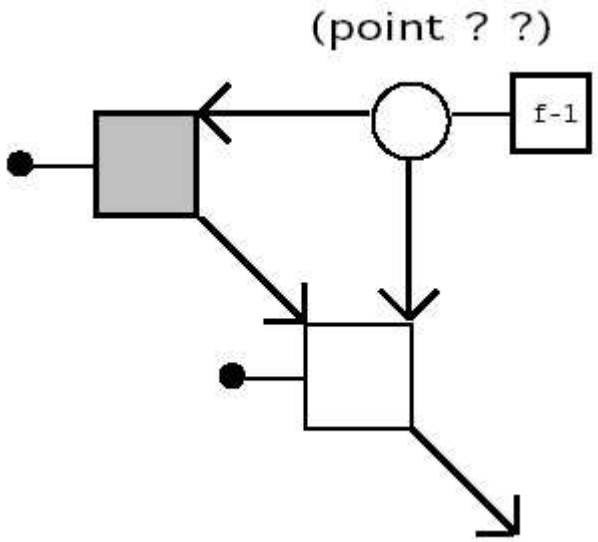

Figura 3.8: Esempio di rete join per una regola di CLIPS. Schema 2

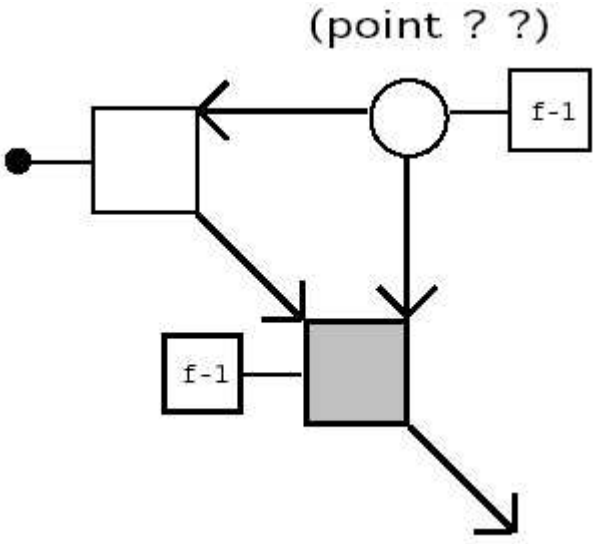

Figura 3.9: Esempio di rete join per una regola di CLIPS. Schema 3

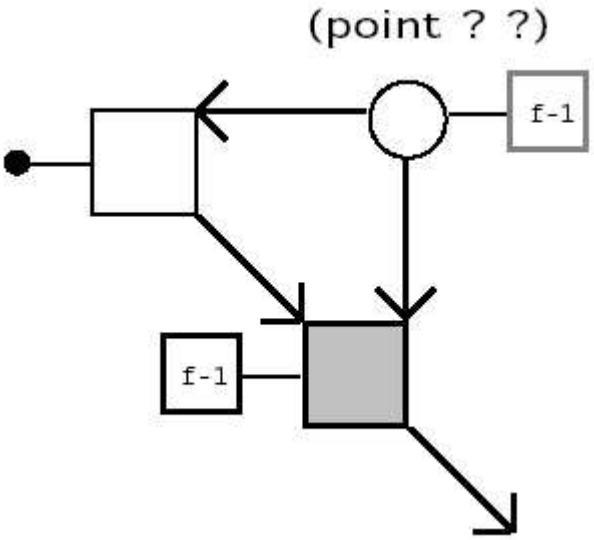

Figura 3.10: Esempio di rete join per una regola di CLIPS. Schema 4

con questo join. In questo momento nella memoria alfa è presente solamente un match parziale. Per il secondo join il match parziale nella memoria alfa è il fatto assegnato al primo pattern ed il match parziale presente nella memoria beta è il fatto assegnato al secondo pattern. La valutazione dell'espressione di join produrrà un risultato FALSE dal momento che il valore assegnato a *?x* nel primo pattern è 3 mentre il valore asegnato a *?x* nel secondo pattern è 4 (vedi figura 3.10).

Il processo di aggiornamento del top join è completo; ora il match parziale nella memoria alfa viene spedito al join in fondo allo schema, come mostrato in figura 3.11.

Una volta ancora viene effettuata una comparazione fra i fatti allo scopo di verificare gli appropriati assegnamenti alle variabili. Come in precedenza anche questa volta il controllo fallisce (vedi figura 3.12).

Supponiamo ora di inserire il fatto

```
f-2  (point 4 3)
```

Questo fatto dovrebbe rispettare il pattern *(point ? ?)* e viene inserito nella memoria alfa del nodo pattern. Il nodo pattern deve quindi passare questo match parziale ai due join ai quali è connesso. Per primo viene spedito al top join come mostrato in figura 3.13.

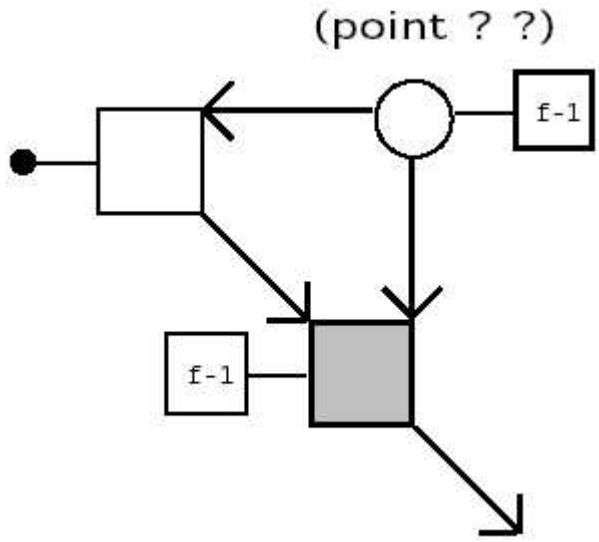

Figura 3.11: Esempio di rete join per una regola di CLIPS. Schema 5

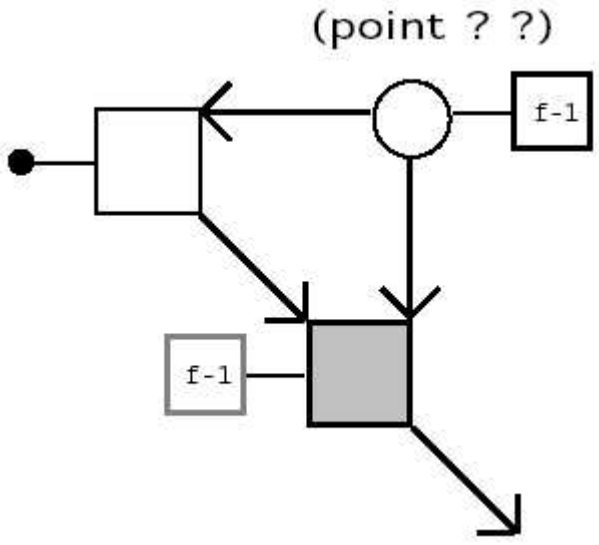

Figura 3.12: Esempio di rete join per una regola di CLIPS. Schema 6

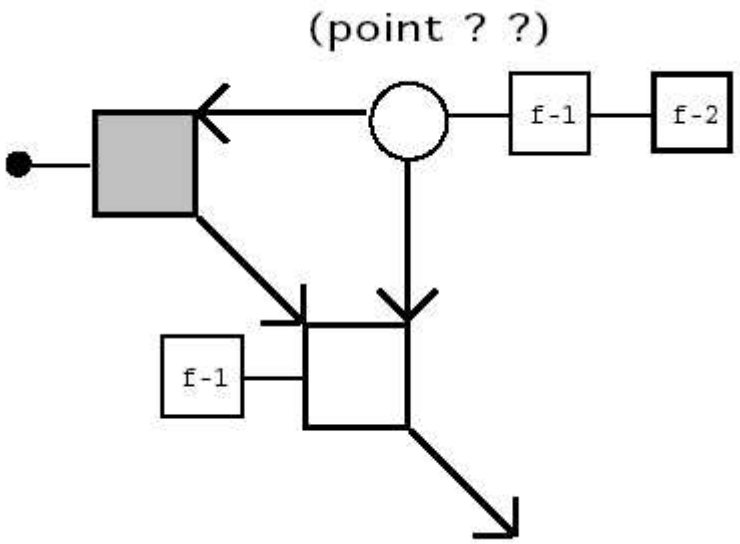

Figura 3.13: Esempio di rete join per una regola di CLIPS. Schema 7

Dal momento che il top join non ha espressioni, il match parziale scende al prossimo join, come mostrato in figura 3.14.

Viene ora effettuata una comparazione tra il match parziale nella memoria beta e tutti i match parziali nella memoria alfa associati a questo join. Il primo confronto avviene fra *f-2* nella memoria beta e *f-1* nella memoria alfa. Questo confronto ha successo. Il valore di *?x* in *f-1* (il secondo campo) è 3 e corrisponde al valore di *?x* in *f-2* (il terzo campo). Viene creato un match parziale consistente in *f-1 f-2*; viene poi spedito al prossimo join. Dal momento che il prossimo join è terminale, la regola che viene soddisfatta e la sua applicazione dovrebbe essere inserita nell'agenda (vedi figura 3.15).

Viene quindi effettuato il prossimo confronto. Il valore di *?x* nel primo pattern (il secondo campo di *f-2*) è 4; il valore di *?x* nel secondo pattern (il terzo campo di *f-2*) è 3. Il confronto fallisce e non vengono creati nuovi match parziali (vedi figura 3.16).

Il processo di aggiornamento del top join è completo. Ora i match parziali presenti nella memoria alfa vengono spediti al join inferiore (vedi figura 3.17).

Viene ora effettuato un confronto fra i match parziali presenti nella memoria alfa associati a questo join e tutti i match parziali nella memoria beta del join. Il primo

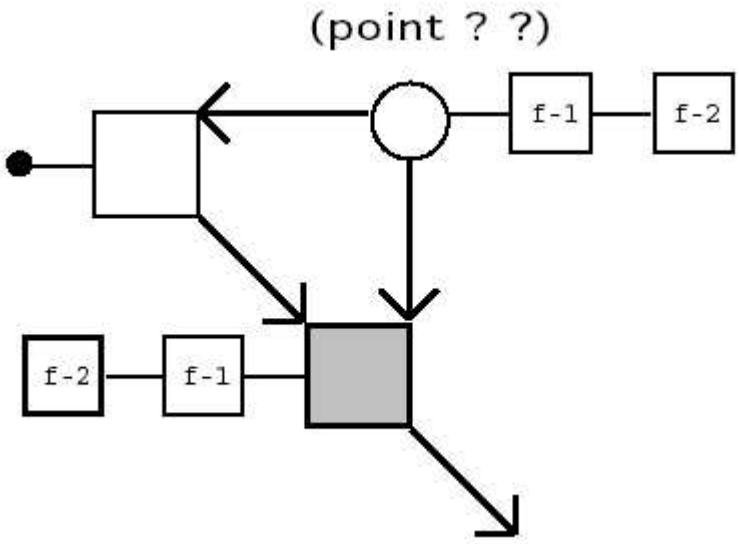

Figura 3.14: Esempio di rete join per una regola di CLIPS. Schema 8

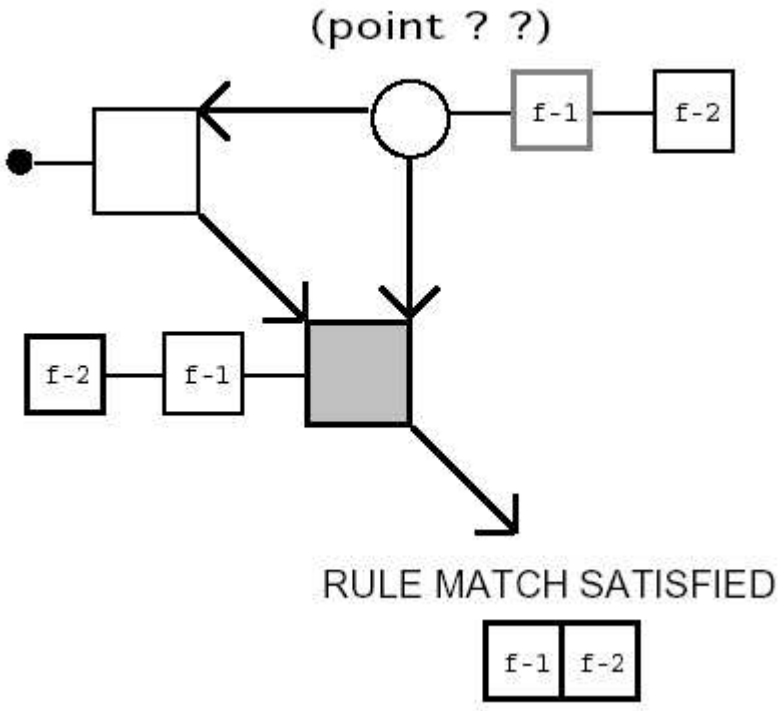

Figura 3.15: Esempio di rete join per una regola di CLIPS. Schema 9

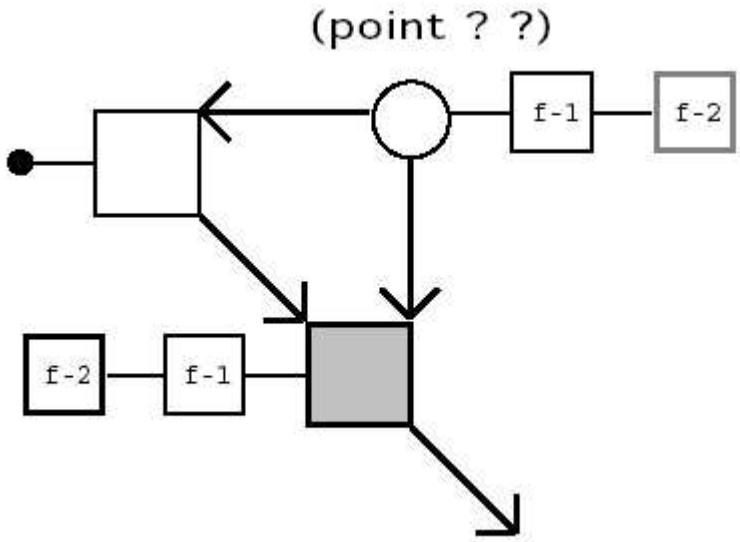

Figura 3.16: Esempio di rete join per una regola di CLIPS. Schema 10

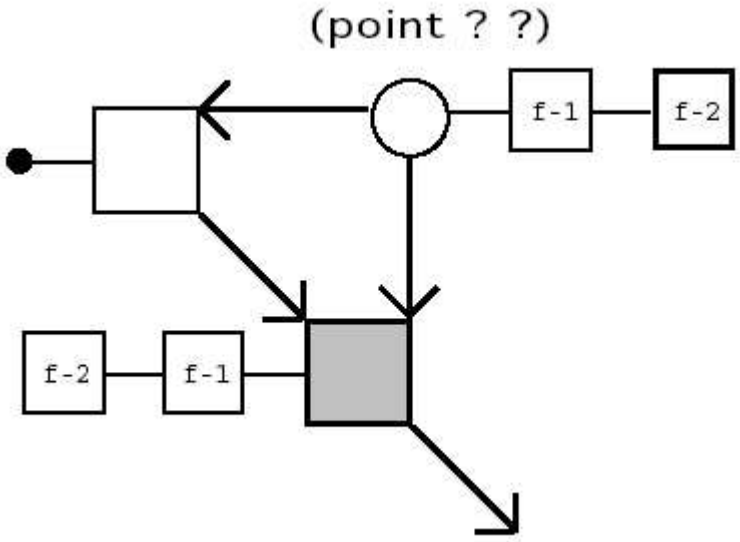

Figura 3.17: Esempio di rete join per una regola di CLIPS. Schema 11

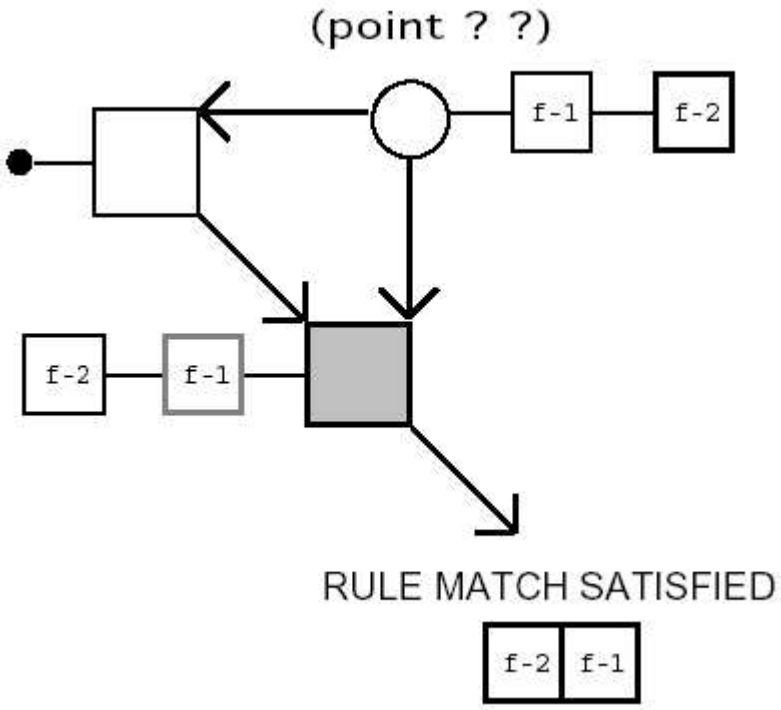

Figura 3.18: Esempio di rete join per una regola di CLIPS. Schema 12

confronto è tra *f-1* nella memoria beta e *f-2* nella memoria alfa. Questo confronto ha esito positivo poiché il valore di *?x* nel primo pattern (il secondo campo di *f-2*) e nel secondo pattern (il terzo campo di *f-1*) è eguale a 4. Viene creato un match parziale costituito da *f-2 f-1* e viene spedito al prossimo join. Poiché quest'ultimo è un join terminale, la regola viene soddisfatta e la sua attivazione dovrebbe essere inserita nell'agenda (vedi figura 3.18).

Viene quindi eseguito il prossimo confronto. Il valore di ?x nel primo pattern (il secondo campo di f-2) è 4 mentre il valore di ?x nel secondo pattern (il terzo campo di f-2) è 3. Il confronto fallisce e non viene creato nessun nuovo match parziale(vedi figura 3.19).

Il processo di aggiornamento del join inferiore è terminato. Da notare il fatto che la memoria beta del top join non ha mai contenuto nessun match parziale.

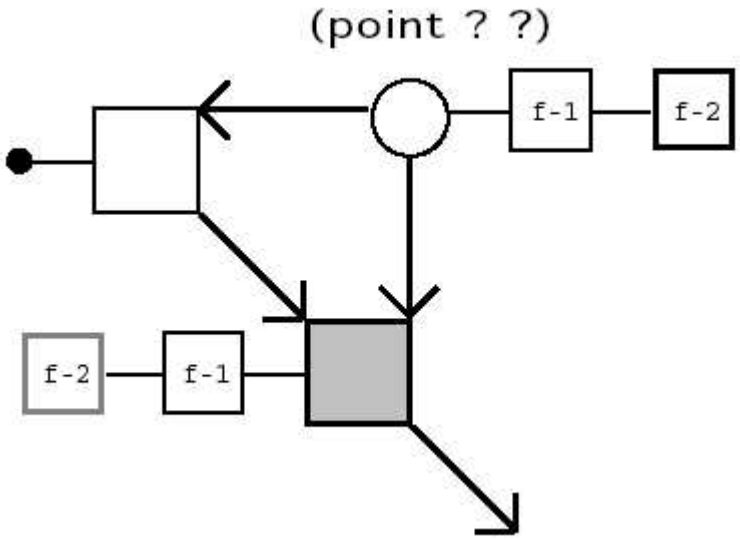

Figura 3.19: Esempio di rete join per una regola di CLIPS. Schema 13

3.3.24 Modulo Match

Il modulo Match contiene le funzioni necessarie per attraversare la pattern network. Questa rete determina quali pattern un fatto ha soddisfatto. La rete pattern è organizzata con una struttura ad albero. I livelli della rete corrispondono direttamente all'ordine sequenziale dei campi trovati nei patterns. Ciò significa che tutti i costrutti pattern trovati nel primo campo di un pattern si trovano nel primo livello della pattern network e così via per i campi successivi. I nodi foglia della rete pattern rappresentano la fine di un pattern. Questi nodi foglia connettono la rete pattern con la rete join.

Ciascun nodo pattern contiene molte informazioni. Innanzitutto si possono trovare i puntatori necessari per costruire la struttura ad albero della rete; vi è poi un'espressione che, quando viene eseguita, è in grado di determinare se un campo ha soddisfatto un costrutto pattern. Inoltre, i nodi foglia contengono un punatore alla memoria alfa dove viene memorizzata una lista di tutti i fatti che soddisfano il pattern; la foglia contiene anche un puntatore alla lista di join nella rete join che sono inseriti dal nodo foglia quando un fatto soddisfa il pattern.

Come esempio consideriamo i seguenti pattern:

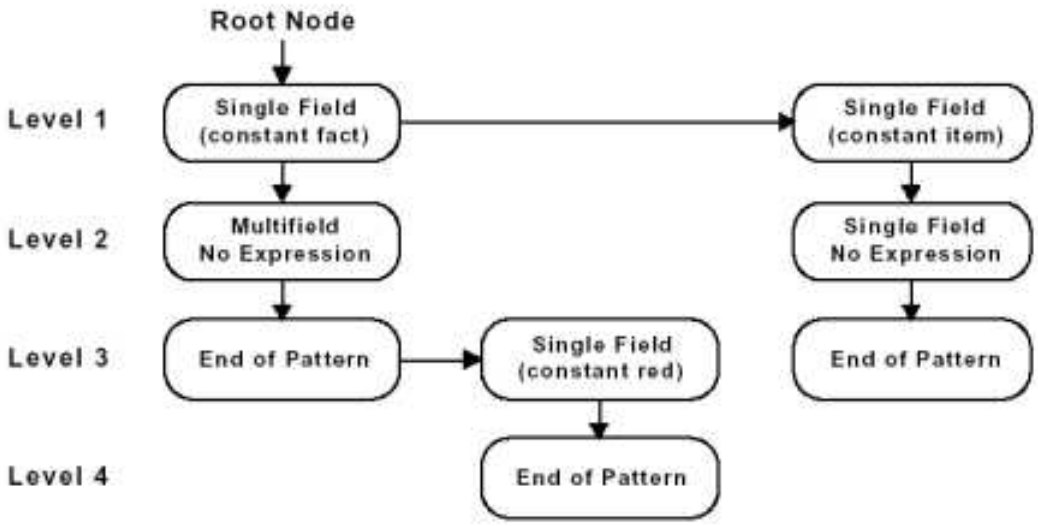

Figura 3.20: Esempio di pattern network

```
(fact $?)
(fact $?x red $?y)
(item ?x)
(item ?y)
```

Essi potrebbero produrre la pattern network in figura 3.20 .

Va notato il fatto che i pattern *(item ?x)* e *(item ?y)* sono trattati come se fossero lo stesso pattern all'interno della pattern network. I pattern *(fact $?)* e *(fact $?x red $?y)* condividono i loro primi due campi. Il nodo "*end of pattern*" più a sinistra è il nodo associato con il match del pattern *(fact $?)* che ha successo. Il modo "*end of pattern*" situato in posizione centrale è il nodo associato al match concluso con successo del pattern *(fact $?x red $?y)*. Il nodo "*end of pattern*" a sinistra è invece associato con la soddisfazione del match con i pattern *(item ?x)* e *(item ?y)*.

Quando un nuovo fatto viene aggiunto alla lista dei fatti deve attraversare completamente la pattern network al fine di determinare quali pattern soddisfa. Come detto, deva l'attraversamento della rete deve essere completo, il ché significa che devono essere trovati tutti i pattern che vengono soddisfatti dal nuovo fatto. L'attraversamento implica testare i campi del fatto con le espressione dei pattern trovati nella rete pattern. Il primo livello della rete effettua test sul primo campo del fatto, il

secondo livello effettua test sul secondo campo del fatto e così via fino a raggiungere l'ultimo livello della rete.

3.3.25 Modulo Retract

Il modulo Retract gestisce le funizonalità principali per l'aggiornamento della rete join quando viene ritrattato un fatto dalla base delle conoscenze.

3.3.26 Modulo Rete Utility

Il modulo Rete Utility contiene le funzioni utili agli altri moduli per implementare l'algoritmo Rete.

3.3.27 Modulo Logical Dependencies

Il modulo Logical Dependencies fornisce le routine di supporto necessarie per l'implementazione dell'elemento condizionale **logical**. Un fatto asserito da una regola senza elementi condizionali **logical** nella LHS della regola è supportato in modo incondizionato. Ciò significa che quel particolare fatto, come tutti i fatti che sono supportati incondizionatamente, può essere ritratto solamente in modo esplicito (cioè non può essere ritratto come risultato diretto della ritrattazione di un altro fatto). I fatti asseriti grazie a deffacts, dal prompt di comando di alto livello o come risultato di azioni accadute al di fuori della portata dell'esecuzione di una regola con elementi condizionali **logical** sono anch'essi supportati in modo incondizionato. Un fatto asserito da una regola con un **logical** CE nella LHS della regola è supportato logicamente da quella regola. Il gruppo di fatti contenuto all'interno del **logical** CE fornisce il supporto logico per i fatti asseriti.

Dal momento che i **logical** CE devono apparire come i primi pattern nella LHS di una regola e che possono non esserci spazi tra elementi condizionali **logical**, esiste un match parziale per la regola contenente tutti i fatti che fornise il supporto logico; questo match parziale è conentuno nella memoria beta di uno dei join della regola stessa. Le dipendenze logiche sono implementate mantenendo due tipi di link. Innanzitutto i link vengono creati tra un match parziale e tutti i fatti che ricevono il supporto logico da quel match parziale. Inoltre, vengono creati link inversi fra i fatti ed il match parziale.

Quando si effettua l'operazione di parsing di una **defrule** che contiene elementi condizionali **logical**, viene calcolata la locazione dei join che conterranno i match parziali che forniranno il supporto logico. Un puntatore a questi join viene salvato come parte integrante della struttura dati della defrule. Quando la regola sarà eseguita, il puntatore al join verrà memorizzato nella variabile globale **TheLogicalJoin**. Le asserzioni che vengono eseguite dalla RHS della regola in esecuzione possono creare i link appropriati tra il match parziale memorizzato nel join referenziato dalla variabile globale ed i fatti a cui fornire supporto logico.

Consideriamo ora questi due esempi che spiegano come vengono utilizzati i link sopra citati per fornire il supporto logico ai fatti:

```
(defrule Example-1
 (logical (a)
          (b))
 (d)
 =>
 (assert (e) (f)))
(defrule Example-2
 (logical (b)
          (c))
 (d)
 =>
 (assert (f) (g)))
```

Assumendo che le regole siano già state caricate, i comandi seguenti eseguirà le regole creando tre nuovi fatti che sono supportati logicamente:

```
CLIPS> (reset)
CLIPS> (assert (a) (b) (c) (d))
CLIPS> (run)
CLIPS>
```

I comandi seguenti illustrano i link di dipendenza logica tra i fatti:

```
CLIPS > (facts)
f-0 (initial-facts)
```

```
f-1 (a)
f-2 (b)
f-3 (c)
f-4 (d)
f-5 (e)
f-6 (f)
f-7 (g)
For a total of 8 facts.
CLIPS> (dependences 5)
f-1,f-3
CLIPS> (dependences 6)
f-2,f-3
f-1,f-2
CLIPS> (dependences 7)
f-2,f-3
CLIPS>
```

I link di supporto logico tra il match parziale ed i fatti che dipendono dal match parziale sono mostrati in figura 3.21.

I fatti *(e)* e *(f)* che vengono asseriti dalla regola *Example-1* sono logicamente dipendenti dal match parziale che contiene i fatti *(a)* e *(b)*. In modo del tutto simile i fatti *(f)* e *(g)* asseriti dalla regola *Example-2* sono logicamente dipendenti dal match parziale che contiene i fatti *(b)* e *(c)*. Bisogna sottolineare il fatto che ciascun match parziale sostiene due fatti distinti.

I link di supporto logico tra fatti ed i match parziali da cui essi ricevono la loro logica sono mostrati in figura 3.22.

3.3.28 Modulo Defrule Manager

Il modulo Defrule Manager contiene un set di funzioni che inizializzano e forniscono supporto ad alto livello per il costrutto defrule.

3.3.29 Modulo Defrule Deployment

Il modulo Defrule Deployment fornisce le funzionalità per implementare le funzioni **bload**, **bsave** e **constructs-to-c** per il costrutto *defrule*.

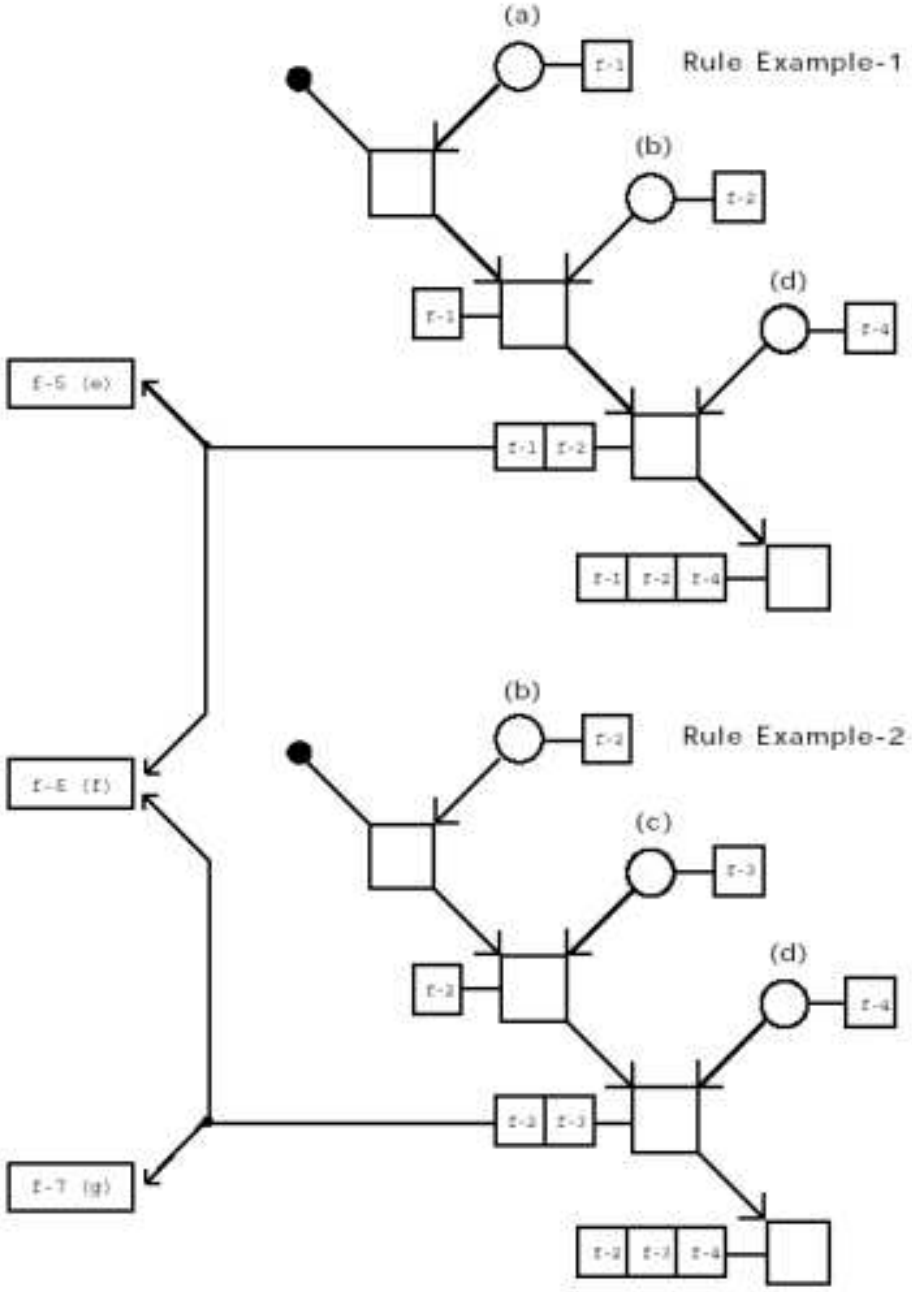

Figura 3.21: Primo schema di supporto logico

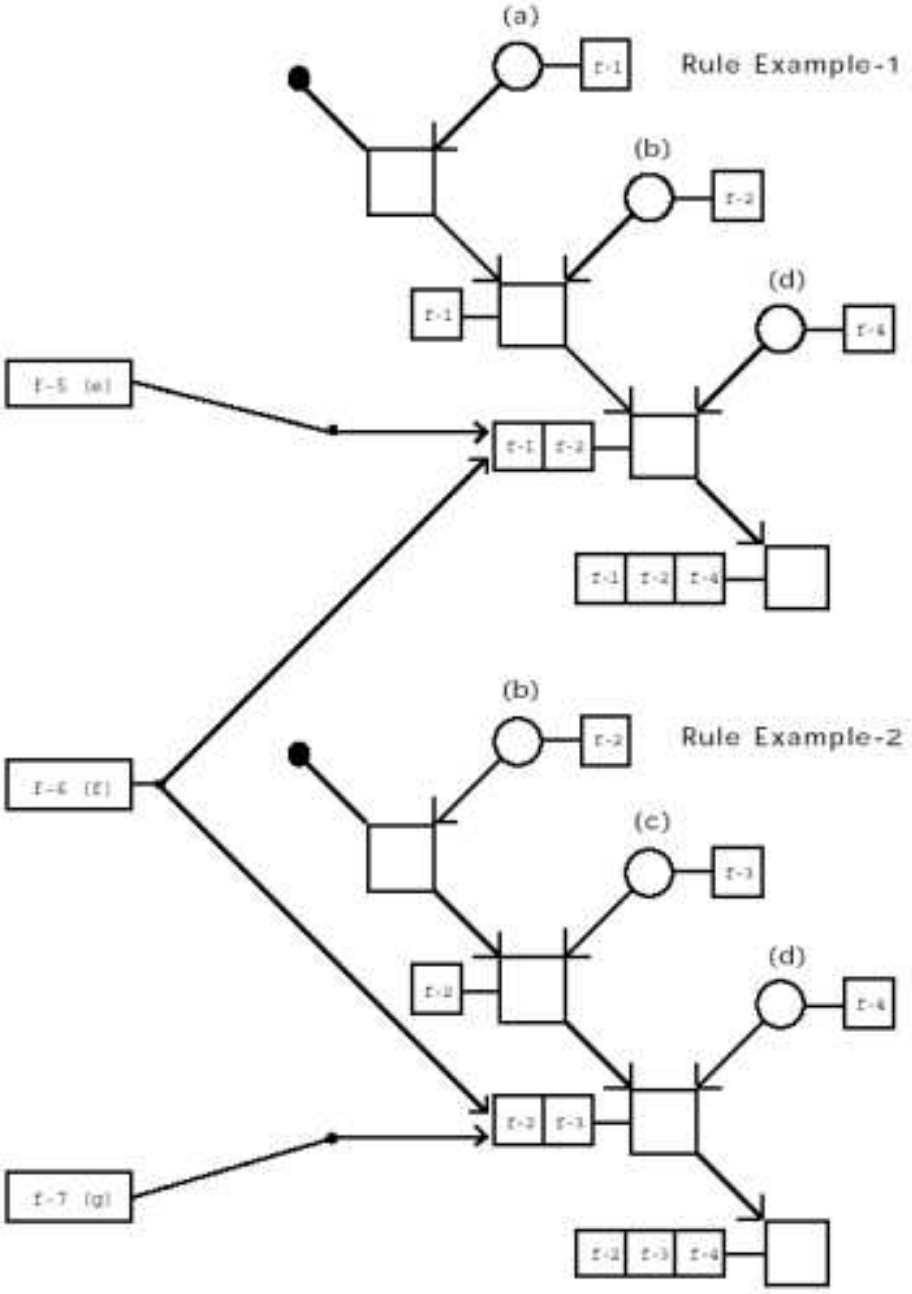

Figura 3.22: Secondo schema di supporto logico

3.3.30 Modulo Defrule Commands

Il modulo Defrule Commands fornisce un gran numero di comandi per manipolare ed esaminare i defrule utilizzati nei programmi CLIPS. I comandi sono: **run**, **undefrule**, **refresh**, **halt**, **rules**, **ppdefrule**, **get-incremental-reset**, **set-incremental-reset**, **set-break**, **remove-break**, **show-breaks**, **matches**, **agenda**, **get-strategy**, **set-strategy**, **get-salience-evaluation**, **set-salience-evaluation** e **refresh-agenda**.

3.3.31 Modulo Deftemplate Commands

Il modulo Deftemplate Commands amministra i comandi associati al costrutto deftemplate. Questi comandi includono **undeftemplate**, **ppdeftemplate**, **list-deftemplate**, **modify**, **duplicate**, **set-dynamic-template-checking** e **get-dynamic-deftemplate-checking**. Questo modulo definisce anche estensioni per i comandi **save**, **clear**, **bload**, **bsave** e **constructs-to-c**.

3.3.32 Modulo Deftemplate Funcions

Il modulo Deftemplate Funcions fornisce routine di parsing per i comandi **modify** e **duplicate**; fornisce inoltre i pattern template utilizzati nel comando **assert**. I pattern template sono convertiti da CLIPS in pattern di tipo regolar position attraverso una serie di semplici regole di traduzione. Usando campi chiave, gli altri campi di un pattern template possono essere specificati in qualsiasi ordine. I campi chiave verranno comunque tradotti in un ordine posizionale fissato.

Consideriamo ora il seguente esempio:

```
(deftemplate example
 (multifield z)
 (field x (default 3))
 (field y))
```

Il pattern RHS usato nel seguente assert:

```
(assert (example (z c d e) (y b) (x a)))
```

dovrebbe essere tradotto in:

```
(assert (example a b c d e))
```

Se un valore non viene specificato in un pattern template, sarà utilizzato un appropriato valore per il pattern. Se non viene specificato un valore di default per un campo, CLIPS fornisce un valore di default basato sui tipi di dato permessi per quel campo e sulla cardinalità del campo stesso (single-field o multifield).

3.3.33 Modulo Deftemplate Parser

Il modulo Deftemplate Parser contiene le funzioni necessarie per effettuare il parsing di un costrutto deftemplate.

3.3.34 Modulo Deftemplate LHS

Il modulo Deftemplate LHS fornisce routine di parsing per i pattern deftemplate trovati nella LHS di una regola.

3.3.35 Modulo Binary Save

Il modulo Binary Save fornisce le funzionalità per il comando **bsave**. Per illustrare come viene effettuato un salvataggio in formato binario viene presentato il seguente esempio:

```
(deffacts start-info
 (point 3.7 5.3))
25
(deffuncion compute ()
 (+ (* 2 3) (* 9.1 2.3)))
```

Le espressioni associate a *start-info* e *compute* sono mostrate, rispettivamente, in figura 3.23 ed in figura 3.24.

Da ora in poi assumeremo che i tipi di dato integer, i puntatori ed i puntatori a numeri di tipo double precision float occupino 4 byte di memoria, mentre i dati di tipo carattere occupino solamente 1 byte.

Il primo step nella creazione di un file binario di immagine è la scrittura di un header binario per il file affinché il comando **bsave** sia in grado di verificare che il file sia un file binario di CLIPS e sia in grado di determinare quale versione di CLIPS abbia creato il file. L'header binario viene scritto in due parti distinte. La prima parte indica che il file è un file binario di CLIPS e contiene caratteri di controllo al

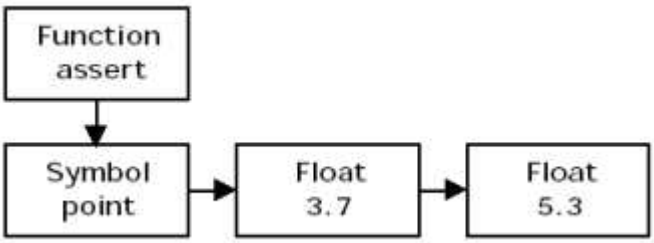

Figura 3.23: Espressione Deffacts *start-info*

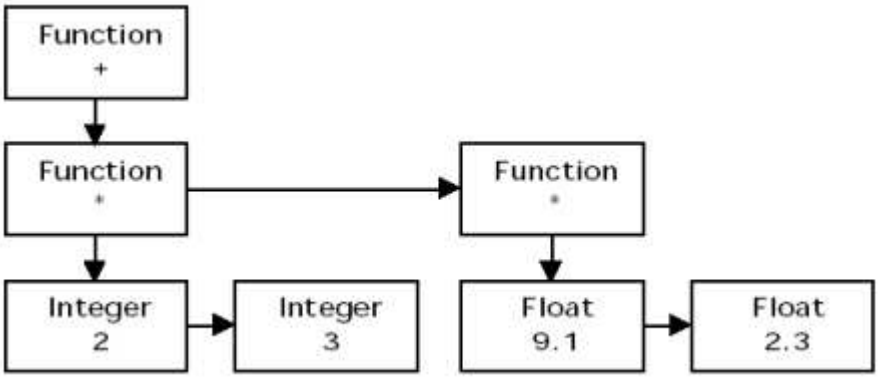

Figura 3.24: Espressione Deffuncion *compute*

fine di aiutare a prevenire l'eventualità che un file testo venga scambiato per un file binario. La seconda parte contiene il numero che specifica la versione di CLIPS che è stata utilizzata per creare il file binario. Un esempio di header binario potrebbe essere:

"\1\2\3\4CLIPS"	"V5.00"

Nel secondo step tutte le funzioni, i simboli, gli integer ed i float all'interno di CLIPS ricevono un flag di stato settato FALSE ad indicare che ciascuno di questi elementi non è necessario al file binario. Quindi, ciascun costrutto che è stato registrato utilizando la funzione **AddBinaryItem** utilizza una chiamata ad una funzione specifica che ha il compito di marcare quali funzioni, simboli, integer e float devono essere salvati nell'immagine binaria poiché necessari al costrutto in esame. Infine, dopo che viene data la possibilità a tutti i costrutti di marcare gli elementi di cui necessitano, viene assegnato un integer id a ciascun item richiesto; tale id verrà utilizzato come riferimento nell'immagine binaria. Ritornando al codice di esempio all'inizio di queta sezione, sono necessarie le funzioni *assert*, + e *; i simboli *start-info*, *compute* e *point*; i float *3.7*, *5.3*, *9.1* e *2.3*; gli interi *2* e *3*.

Nel terzo step, ciascuna delle funzioni richieste viene scritta nell'immagine binaria. Viene scritto il numero di queste funzioni, seguito dall'ammontare dello spazio richiesto per il nome delle funzioni e dal nume di ogniuna delle funzioni sotto forma di una stringa terninata da NULL. **òjkldfsjklsdfòjklfasdlòjksfd**. Bisogna notare che queste ultime non vengono scritte utilizzando un ordine specifico; una funzione è scritta una sola volta (indipendentemente dal numero di item binari che la utilizzano).

Riprendendo l'esempio precedente, questo dovrebbe essere l'output scritto nel file binario:

3	4 byte: Numero di Funzioni
11	4 byte: Dimensione Totale dei Nomi delle Funzoni
"+"	2 byte: Spazio della Funzione #0
"*"	2 byte: Spazio della Funzione #1
"assert"	2 byte: Spazio della Funzione #2

Nel quarto step ciascun simbolo che è stato identificato come necessario viene scritto nell'imagine binaria. Vengono scritti il numero di simboli necessari, la dimensione dello spazio occupato da tutti i nomi dei simboli, i singoli nomi dei simboli sottoforma di stringhe terminate da NULL. Continuando con l'esempio precedente abbiamo:

3	4 byte: Numero dei Simboli
26	4 byte: Dimensione Totale dei simboli
"start-info"	12 byte: Spazio del Simbolo #0
"compute"	8 byte: Spazio del Simbolo #1
"point"	6 byte: Spazio del Simbolo #2

Nel quinto step ciascun float ritenuto necessario viene scritto nell'immagine binaria. Viene scritto il numero dei float necessari seguito dai float stessi, come mostrato qui di seguito:

4	4 byte: Numero di Float
9.1	8 byte: Spazio del Float #0
2.3	8 byte: Spazio del Float #1
3.7	8 byte: Spazio del Float #3
5.3	8 byte: Spazio del Float #4

Nel sesto step vengono scritti all'interno dell'immagine binaria tutti gli ingeger ritenuti necessari. Continuando l'esempio iniziale avremo:

4	4 byte: Numero di Integer
2	4 byte: Spazio dell'Integer #0
3	4 byte: Spazio dell'integer #1

Nel settimo step nell'immagine binaria viene scritta ciascuna delle espressioni necessaria ai costrutti. Innanzitutto viene scritto il numero totale delle espressioni, quindi viene chiamato ciascun costrutto registrato il modo che inserisca nel file binario le proprie espressioni. Ciascuna espressione viene inserita nel file binario utlizzando il seguente formato:

10				4 bytes: Number of Expressions
FUNCTION	2	1	-1	16 bytes: Function assert #0
SYMBOL	2	-1	2	16 bytes: Symbol point #1
FLOAT	2	-1	3	16 bytes: Float 3.7 #2
FLOAT	3	-1	-1	16 bytes: Float 5.3 #3
FUNCTION	0	5	-1	16 bytes: Function + #4
FUNCTION	1	6	8	16 bytes: Function * #5
INTEGER	0	-1	7	16 bytes: Integer 2 #6
INTEGER	1	-1	-1	16 bytes: Integer 3 #7
FUNCTION	1	9	-1	16 bytes: Function * #8
FLOAT	0	-1	10	16 bytes: Float 9.1 #9
FLOAT	1	-1	-1	16 bytes: Float 2.3 #10

Figura 3.25: Contenuto del file binario creato con la funzione **bsave**

TYPE	INDEX VALUE	ARGUMENT LIST	NEXT ARGUMENT

Il campo *type* corrisponde direttamente al valore dello stesso campo nella struttura dati dell'espressione di CLIPS. I campi *index value*, *argument list* e *next argument* sono dei puntatori nella struttura dati di CLIPS, ma vengono convertiti in indici integer quando vengono scritti nell'immagine binaria. Per esempio, l'*index value* per il simbolo *point* dovrebbe essere 2 in quanto è il terzo item salvato nella sezione symbol dell'immagine binaria (bisogna ricordare che l'ennesimo elemento in un vettore C è referenziato da n-1). Anche i puntatori alle altre espressioni o alle altre strutture dati sono trasformati in integer allo stesso modo. Un puntatore NULL viene convertito nel valore -1. Considerando l'esempio precedente, nel file binario avremo i dati mostrati in figura 3.25.

Nell'ottavo step ciascun costrutto registrato viene scritto nell'immagine binaria. Innanzitutto viene scritto il nome del costrutto (vi è un mumero massimo di caratteri a disposizione). Viene quindi chiamata la funzione **bsave** per il costrutto. Questa funzione scrive nel file immagine la dimensione richiesta per inserire nel file

il costrutto, quindi passa a scrivere nell'immagine binaria qualsiasi struttura dati di cui il costrutto ha bisogno.

3.3.36 Modulo Binary Load

Il modulo Binary Load fornisce le funzionalità per il comando **bload**. Di seguito vengono descritti tutti gli step che permettono di caricare un'immagine binaria.

Il primo step consiste nel caricamento del binary header dal file, al fine di determinare se il file sia davvero un'immagine binaria e da quale versione di CLIPS sia stata creata. Quindi viene pulito l'ambiente CLIPS (tramite un processo simile a quello generato dal comando **clear**). Vengono quindi caricate le funzioni presenti all'interno dell'immagine binaria. Se qualcuna di queste funzioni non è disponibile l'intero processo di caricamento dal file binario viene abortito. Ciò significa che se l'immagine binaria è stata creata da una versione modificata di CLIPS nella quale l'utente ha creato nuove funzioni, questa immagine è caricabile in versioni standard di CLIPS o in versioni modificate da altri utenti, a patto che non vengano utilizzate le nuove funzioni definite dall'utente che ha creato il file binario. Dopo che sono state caricate la funzioni necessarie, vengono caricate dall'immagine i simboli, i float e gli integer seguiti da tutte le espressioni necessarie.

Terminato il caricamento degli item fomdamentali si passa alle informazioni relative ai costrutti. Viene prima caricato il nome del costrutto, quindi vengono caricate le funzioni appropriate per il costrutto registrato al fine di poterlo caricare dall'immagine binaria. Se il costrutto non è registrato, le informazioni ad esso associate vengono saltate durante il caricamento dal file binaro; ciò permette di poter caricare un file binario generato da una versione modificata di CLIPS all'interno di una versione standard o di un'altra versione modificata. Il processo di lettura delle informazioni relative ai costrutti si ripete fino a raggiungere la fine del file binario.

3.3.37 Modulo Construct Compiler

Il modulo Construct Compiler fornisce le funzionalità che permettono al comando **constructs-to-c** di generare il codice C rappresentante i costrutti presenti in quel momento nell'ambiente CLIPS. I file consì generati possono essere compilati e linkati ad una versione di CLIPS runtime specificatamente compilata per poter utilizzare i

file stessi. Il vantaggio di questo modo di operare sta' nelle minori dimensioni della verisone di CLIPS runtime rispetto alle dimensioni della versione standard.

Il compilatore dei costrutti lavora in modo simile alle operazioni **bsave** e **bload**, anche se i costrutti non vengono inseriti in un file binario. Dal momento che il file oggetto creato da questo modulo può essere linkato direttamente a CLIPS, non vi è la necessità di caricare i costrutti all'interno dell'ambiente del sistema esperto.

Consideriamo ora un esempio che ci permetterà di capire come funziona il compilatore dei costrutti. Assumiamo dunque che il seguente deffacts sia stato inserito all'interno della base delle conoscenze:

```
(deffacts start-info
 (point 3.7 5.3))
```

Inoltre, nella base delle conoscenze viene inserito automaticamente il seguente deffacts:

```
(deffacts initial-fact
 (initial-fact))
```

Supponiamo ora che l'utente abbia inserito il seguente comando nella console:

```
CLIPS> (constructs-to-c xmp 3)
CLIPS>
```

Il primo step seguito dal compilatore dei costrutti è quello di generare un header file che sarà usato dal file sorgente C che sarà generato. Poiché l'utente ha specificato come simbolo prefissato *xmp*, quindi verrà generato un header file chiamato "xmp.h". Inizialmente in questo file vengono scritte tutte le definizioni esterne delle funzioni di sistema e di quelle definite dall'utente. In un secondo momento, se necessario, in questo file potranno essere scritte anche le definizioni di strutture dati in modo da permettere ad altri file di poterle utilizzare.

Si procede quindi a scrivere all'interno del file tutte le definizioni dei simboli, dei float, degli integer e delle funzioni. Quando tutti i tipi di dato di base sono stati scritti nel file, ciascun costrutto che è stato registrato dal compilatore dei costrutti, attraverso la funzione **AddCodeGeneratorItem**, viene chiamato in modo che possa generare il proprio codice C. Ciascun costrutto è responsabile della generazione del codice relativo alle strutture dati che ha definito. I riferimenti ai simboli, ai float

ed agli integer sono risolti attraverso la chiamata alle funzioni **PrintSymnbolReference**, **PrintFloatReference** e **PrintIntegerReference** che inseriranno nel file C generato dal costrutto i riferimenti appropriati ai data item. Ciascun costrutto deve effettuare la chiamata alla funzione **ExpressionToCode** per qualsiasi espressione che deve essere salvata.

Capitolo 4

Intrusion Detection

Gli Intrusion Detection System (IDS) sono sistemi informatici che rappresentano un sistema d'allarme per un sistema informatico. In generale, infatti, questi sistemi si occupano di identificiare e rilevare gli attacchi a cui è sottoposto un sistema informatico. Per fare ciò estraggono informazioni da un insieme di sistemi o risorse della rete, le analizzano e tentano di identificare, preferibilmente in tempo reale, se vi è utilizzo non autorizzato delle risorse, da parte di utenti interni o esterni alla rete controllata. In generale sono strumenti di rilevazione e non di prevenzione.

Fondamentale è la tempestività dell'intervento: appena rilevato l'attacco, il sistema deve avvertire il responsabile della sicurezza ed eventualmente agire in maniera automatica per contrastare l'attacco rilevato, anche se non tutti gli IDS hanno quest'ultima funzionalità. Solitamente hanno una struttura modulare in cui ogni componente ha un compito specifico: gestione degli eventi sospetti, contromisure, immagazzinamento sicuro dei dati di log, etc.

La descrizione più generale di un IDS è quella fornita dalla Common Intrusion Detection Framework [CIDF]. Il modello in figura 4.1 è riferito ai Network IDS, ma può essere facilmente esteso ad altre classi di IDS.

Andiamo a descrivere in dettaglio i quattro sottoinsiemi presentati nel modello:

1. *Passive-Protocol Analyser* (E-box): esegue la cattura dei pacchetti dalla rete. Ogni pacchetto viene elaborato come se fosse arrivato al suo rela host di destinazione. Questo box è in pratica un protocol stack particolare, nel senso che deve solamente emulare il comportamento del protocol stack della maggior

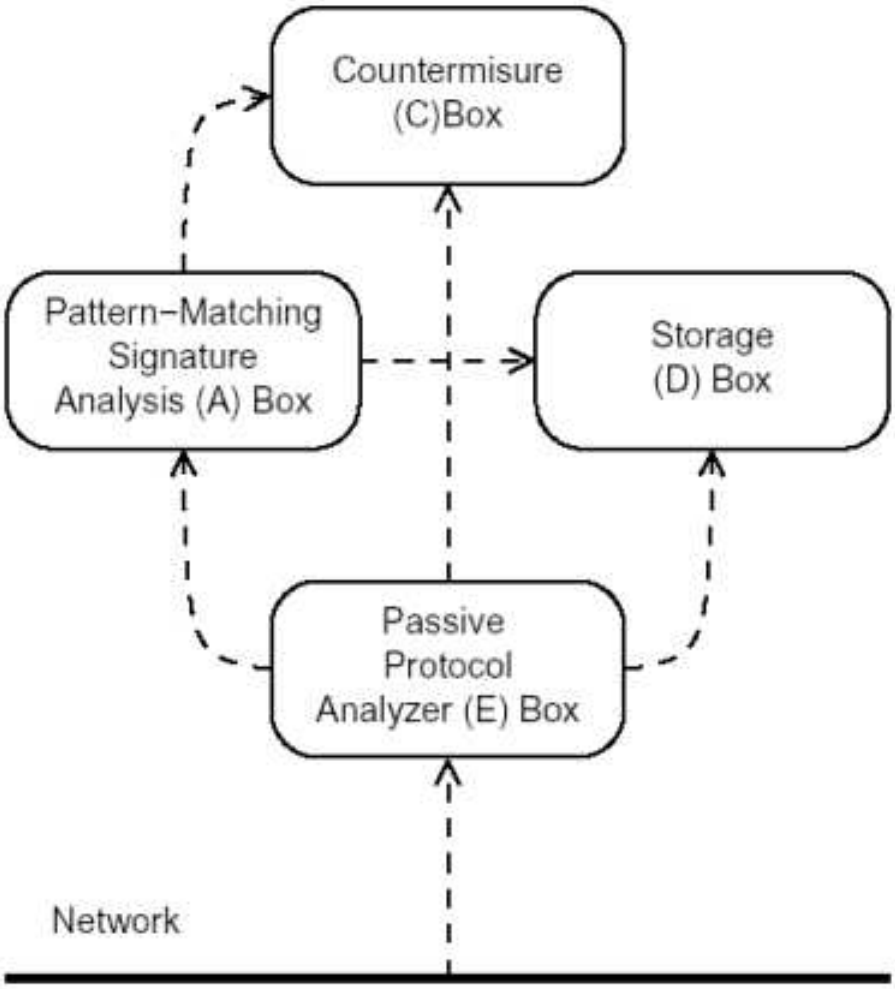

Figura 4.1: Struttura di un Network IDS

parte dei sistemi operativi in commercio in modo da rimanere sincronizzato con essi. Le operazioni elementari che deve eseguire sono:

- Verificare la correttezza del pacchetto (checksum, sorgente ≠destinazione, etc.)
- Deframmentazione del pacchetto IP
- Riassemblamento dell stream TCP

A questo livello, è già possibile effettuare l'Intrusion Detecion poiché osservando pacchetti troppo frammentati e troppo fuori sequenza, o che non seguono gli standard di protocollo è possibile ipotizzare un tentativo di attacco, oppure un malfunzionamento di qualche dispositivo di rete.

2. *Pattern-Matching Signature Analysis* (A-box): esegue l'analisi del payload del singolo pacchetto. i primi modelli di IDS si limitavano al pacchetto, mentre i nuovi modelli eseguono anche l'analisi delle anomalie di protocollo (ad esempio osservano lo stream di un'intera sessione HTTP). La ricerca di un pattern all'interno del payload è giustificata dal fatto ceh gran parte degli attacchi visibili in rete contengono delle signature (delle firme caratteristiche) standard dal momento che gran parte degli attacchi sono effettuati con strumenti automatizzati. Questo, però, non si verifica per attacchi 0-day, cioè ancora sconosciuti. Per questo motivo l?A-box viene integrato con l'analisi di protocollo che permette di identificare anomalie e quindi tentativi di exploit delle debolezze.

3. *Storage Box* (D-box): la grande quantità di dati prodotta dall'E-box e dall'A-box sonon raccolte dal D-box. La memorizzazione delle informazioni è necessaria per i seguenti motivi:

   - Gli allarmi memorizzati sono visibili e controllabili anche dopo molto tempo dall'attacco che li ha generati
   - Per ogni attacco effettuato dovrebbe essere possibile analizzare i dati delle comunicazioni contestuali
   - Avere i dati sul traffico catturato permette di effettuare operazioni di data-mining: è possibile ottenere informazioni sia su problemi di perfrormance della rete che su andamenti anomali, sintomo di un tentativo di attacco o di un attacco già riuscito.

- Analisi legale: la raccolta di informazioni permette di presentare delle prove sui reati che sono stati commessi e quindi possono essere richiesti per scopi assicurativi o per scopi processuali.

4. *Countermeasure Box* (C-box): alcuni IDS hanno dei meccanismi di risposta verso i tentativi di intrusione. Uno dei meccamismi più utilizzati per i Network IDS è quello di terminare una comunicazione giudicata ostile. Un altro metodo è quello di effettuare una riconfigurazione del firewall in modo da fargli bloccare determinati indirizzi sorgente. E' stato dimostrato che i due metodi non sono particolarmente efficienti poiché è possibile eluderli oppure utilizzarli per ottenere un attacco DoS contro il sistema. Inoltre bisogna considerare il fatto che vi deve essere una limitazione al numero di regole presenti nel firewall; tale limitazione è dovuta al fatto che ogni regola necessita di una certa quantità di memoria e di un determinato tempo di esecuzione. Se il numero di regole eccede un determinato limite (imposto dalle caratteristiche prestazionali del firewall) si avrà un aumento esponenziale del ritardo con cui il firewall esegue le regole e quindi una totale perdita della sua funzionalità più importante: la tempestività di intervento. In figura 4.2 sono mostrati gli effetti che un eccessivo numero di regole ha sull'efficienza di un firewall [FIR].

Gli IDS si dividono in due grandi categorie:

- **Network Intrusion Detection System** (NIDS): effettuano l'analisi del traffico di rete. Per fare ciò utilizzano le Network Interface Card (NIC) in modalità *promiscuos mode* in modo da poter analizzare tutto il traffico presente sulla rete. Le informazioni lette dalla rete in *promiscuos mode* comprendono tutti i pacchetti, compresi quelli che normalmente vengono filtrati dai dispositivi hardware delle NIC. In base ai metodi di riconoscimento di attacco, i NIDS possono generare un allarme se osservano comunicazioni dal contenuto ostile (buffer overflow), anomalie nei protocolli utilizzati (directory traversal), oppure dei discostamenti dai comportamenti giudicati "normali". In figura 4.3 viene rappresentata la struttura generica di un NIDS.

- **Host based Intrusion Detection System** (HIDS): è un software che risiede su un host e che controlla continuamente lo stato del sistema. Lo stato viene

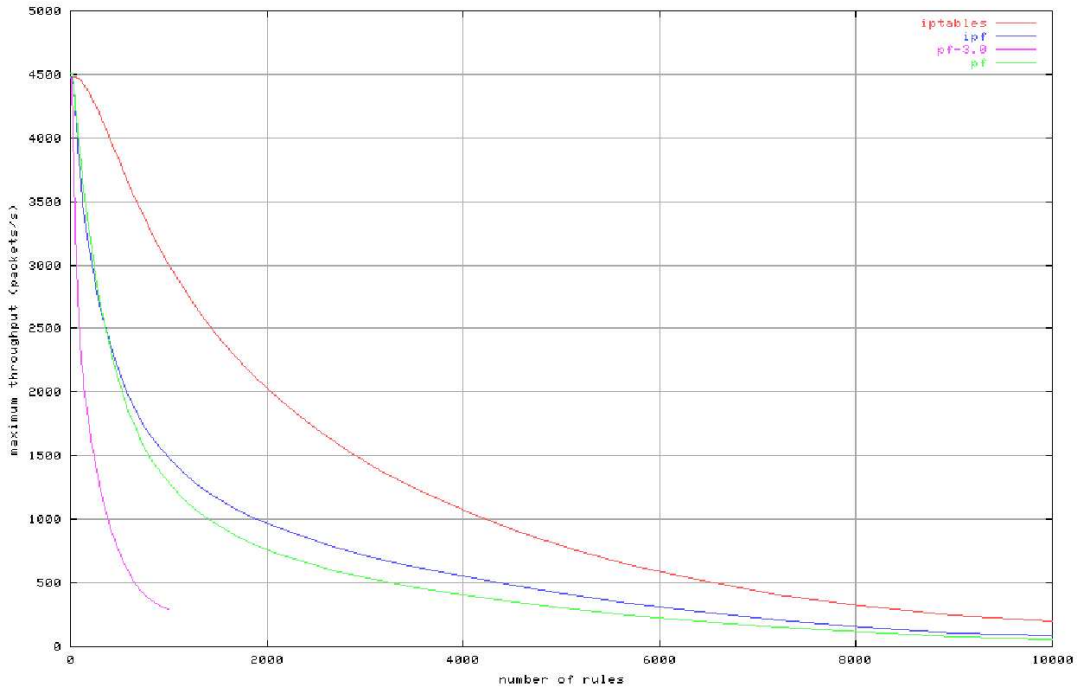

Figura 4.2: Prestazioni dei Firewall al crescere del numero di regole

identificato osservando i log prodotto dalle varie applicazioni in funzione e dall'accounting delle risorse (CPU, Memoria, Disco). Gli HIDS possono avere un comportamento reattivo, quando generano un allarme per notificare che qualcosa è successo, oppure possono essere proattivi, impedendo cioè all'utente di effettuare operazioni pericolose.

Uno degli esempi più diffusi di HIDS proattivo è LIDS (Linux IDS). LIDS è un modulo da installare sul kernel Linux che permette di modificare il sistema di gestione della sicurezza. In pratica viene separata la figura dell'utente root da quella dell'amministratore della sicurezza. L'amministratore della sicurezza può concedere i privilegi per determinate operazioni e può farlo solo operando fisicamente sulla macchina. L'utente root possiede i privilegi forniti dall'amministratore della sicurezza. La struttura generale di un HIDS è mostrata in figura 4.4.

Esistono molte varianti di HIDS. L'application HIDS esegue il monitoring di una singola applicazione (ad esempio un Web Server). I File Integrity Checkers

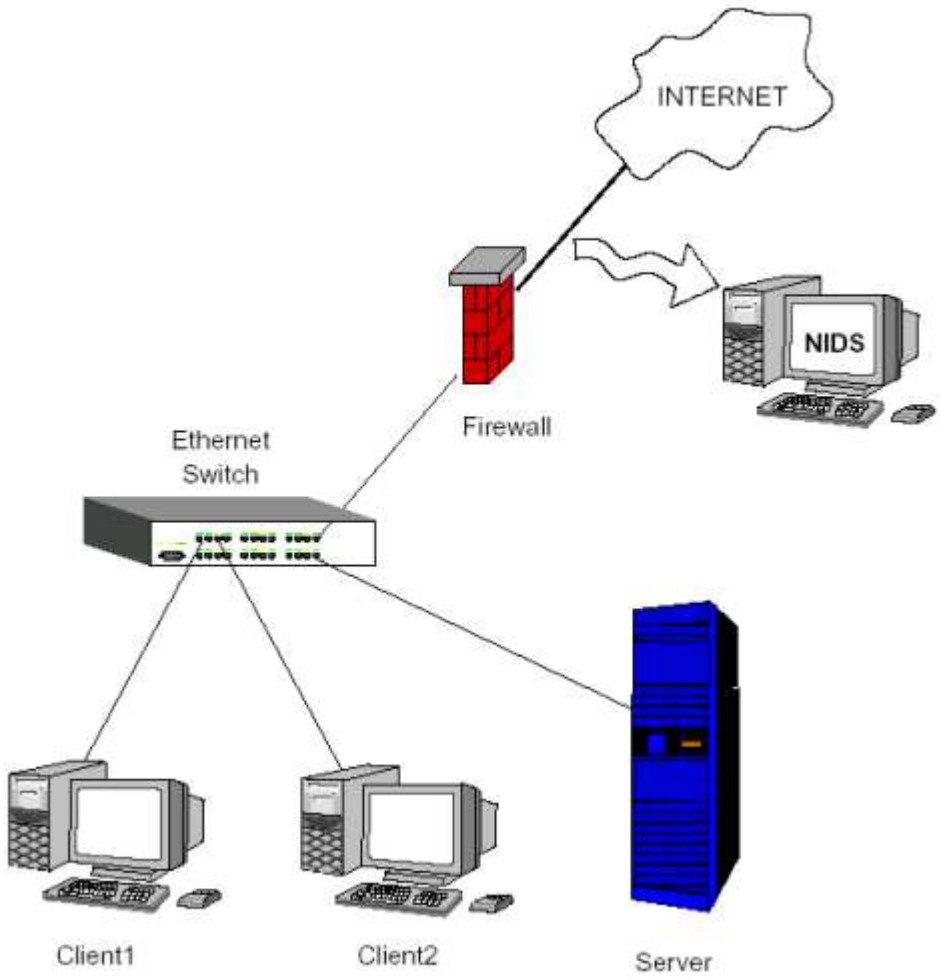

Figura 4.3: Struttura generica di un NIDS

nono HIDS che controllano se un file è stato modificato. I Target based HIDS controllano lo stato di particolari risorse (ad esempio lo spazio su disco oppure l'accesso a particolari file).

- **Sistemi di rilevazione ibridi NIDS + HIDS**: con il termine Hybrid IDS si intendono sistemi composti da 3 blocchi fondamentali:
  - Sensore NIDS
  - Sensore HIDS
  - Un sistema centralizzato che raccoglie gli allarmi dei due sensori e li archivia in un database

  Questo tipo di architettura permette di fondere i vantaggi dei NIDS e HIDS, creando un sistema molto più efficiente e completo, in quanto può rilevare una più grande casistica di attacchi, permettendo l'archiviazione di informazioni inerenti a nuvi attacchi o anomalie. Uno schema gererico di questo tipo di IDS è mostrato in figura 4.5.

4.1 Prelude: Hybrid Intrusion Detection system

Come sarà ampiamente descritto nel capitolo 5, l'oggetto di questa tesi è l'integrazione in un IDS delle potenzialità derivate dalla tecnologia dei sistemi esperti (vedi capitolo 2). Tra tutti i istemi di Intrusion Detection si è scelto il sistema PRELUDE, un IDS ibrido che sfrutta sia le potenzialità dei Network IDS che degli Host based IDS. L'architettura generale di Prelude IDS è mostrata in figura 4.6.

4.1.1 Blocchi fondamentali

Come ogni sistema di rilevamento delle intrusioni, anche Prelude IDS si compone principalmente di tre blocchi distinti:

1. **Sensori**: sono entità di rilevazione situate nelle posizioni strategiche della rete. I sensori sono in grado di segnalare al manater gli allarmi relativi alla sicurezza in maniera particolarmente dettagliata, offrendo un grado molto elevato di informazioni inerenti gli eventi registrati.

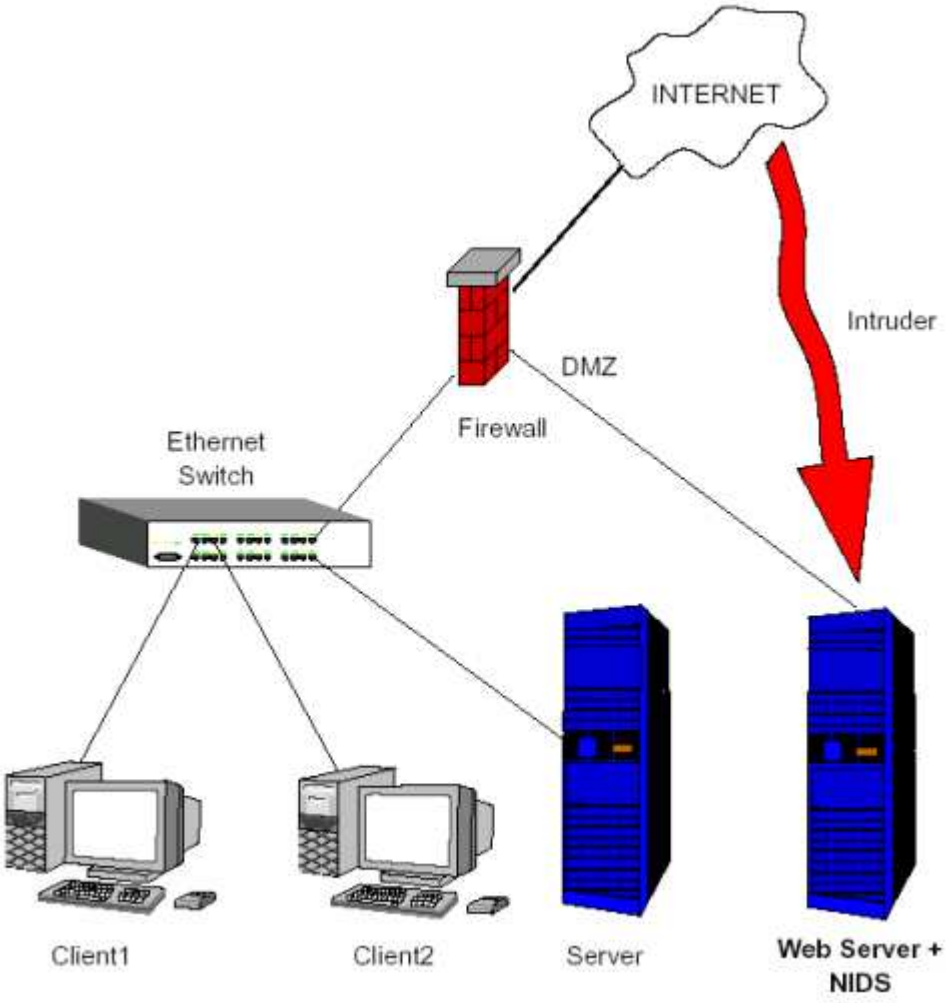

Figura 4.4: Struttura generale di un HIDS

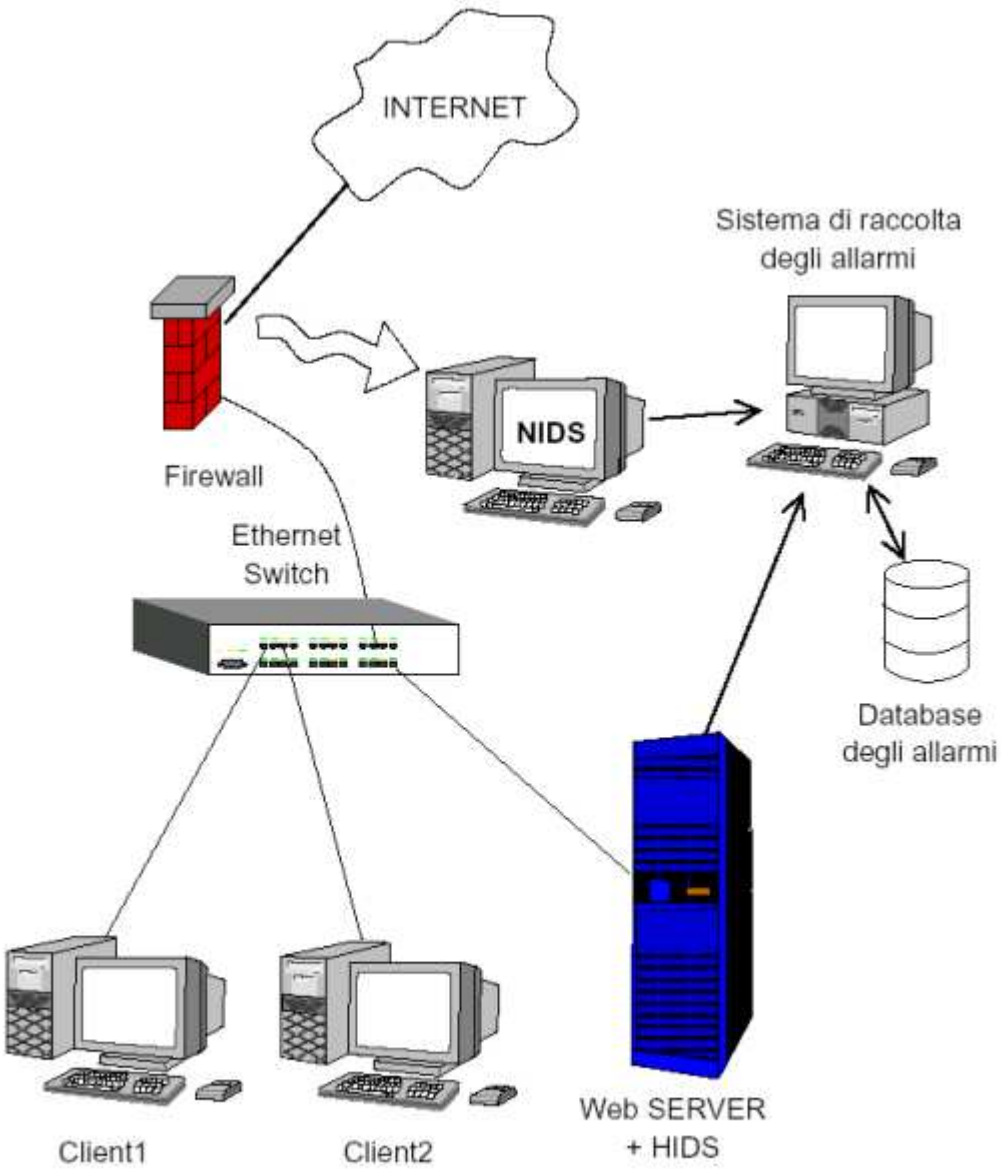

Figura 4.5: Struttura generica di un Hybrid IDS

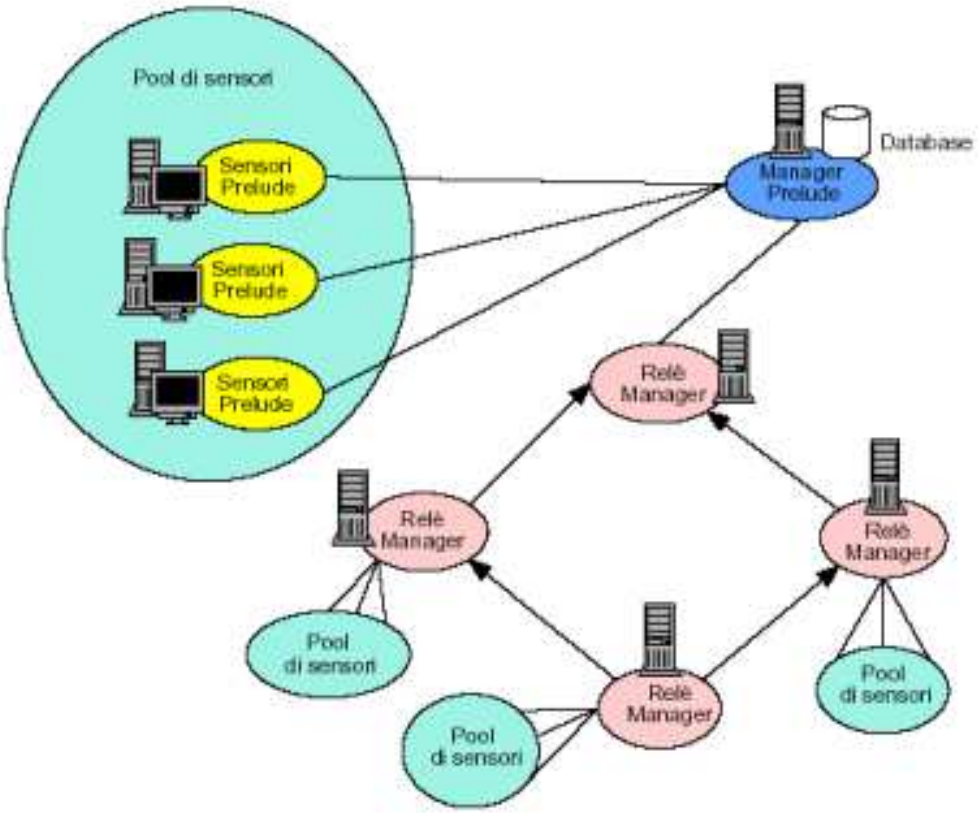

Figura 4.6: Architettura generica di Prelude IDS

2. **Manager**: accettano le connessioni sicure che vengono dai sensori e salvano gli allarmi che i sensori emettono. Le due caratteristiche principali di un manager sono:

   (a) *Logging*: il manager (possono essere uno o più di uno) usa i plug-in (oggetti comuni che possono essere caricati dinamicamente) per convertire gli allarmi dalla disposizione binaria IDMEF (vedi sezione 4.2) nei vari formati di output.

   (b) *Contromisure*: il manager può essere autorizzato dall'utente ad effettuare contromisure a fronte di un attacco (questo è ciò che avviene nella versione di Prelude IDS manager modificata nell'ambito di questa tesi, come illustrato nel capitolo 5). Qualora vi siano più manager nella rete monitorata, le contromisure possono riguardare solamente una parte della rete, in particolare quella controllata dal manager che ha avviato le procedure di reazione.

3. **Agenti per le contromisure**: sono agenti generici fatti funzionare sulle macchine che devono reagire in caso di attacco. Ciascuno di questi agenti ha parecchi plug-in, ogni plug-in ha un driver per ogni genere di contromisure software/hardware (per esempio un collegamento per Netfilter, IPFilter). E' anche possibile l'integrazione di plig-in avanzati per le contromisure che possono fare, per esempio, azioni come l'islanding (per isolare una macchina attaccata) o thotting (per limitare la larghezza di banda utilizzata da un hacker). Durante la fase di programmazione delle contromisure da contrapporre ai vari tipi di attacchi a cui la rete potrebbe essere sottoposta, non si deve mai dimenticare l'importanza che la rete possiede: è infatti molto sottile il confine che, nell'ambito informatico, separa la legittima difesa da una reazione del tutto illegittima.

4. **Frontend**: analizzare gli allarmi ricevuti rappresenta il fatto decisivo che permette di capire cosa stia succedento all'interno della rete. Per le sue caratteristiche, questa analisi non può essere completamente automatizzata e, solitamente, richiede un notevole sforzo da parte dell'amministratore della sicurezza e dei suoi collaboratori. Per aiutare gli amministratori in questo compito viene for-

nito loro un frontend che permette di passare in rassegna tutti gli allarmi e fornisce statistiche con l'obiettivo di risolvere i problemi inerenti la sicurezza.

Come mostrato in figura 4.6, il sistema Prelude IDS prevede la possibilità di installare all'interno della propria LAN più sensori e più manager con un'architettura di tipo distribuito. Per comunicare fra loro, i sensori ed i manager utilizzano un protocollo di comunicazione basato sul recente standard IDMEF (Intrusion Detection Message Exchange Format) [IDMEF] descritto nella sezione 4.2.

Questo formato, basato sul linguaggio XML, è abbastanza generico da permettere che i componenti ibridi di Prelude trasmettano gli allarmi qualificati, specificando qualsiasi informazione utile riguardante gli eventi registrati. il formato IDMEF garantisce una buona interoperabilità, anche se per poter ottimizzare le prestazioni di Prelude IDS gli sviluppatori hanno dovuto costruire un meccanismo di conversione in formato binario dei messaggi IDMEF che entra in gioco prima che questi ultimi vengano spediti. Questa conversione si è resa necessaria poiché l'utilizzo diretto di XML avrebbe richiesto la conversione delle intestazioni binarie in formato testo (per esempio per gli indirizzi IP) e questo avrebbe comportato un notevole tempo di CPU.

Tutte le comunicazioni fra le varie componenti distribuite (fra sensori e manager e fra manager) avviene tramite protocollo SSL con l'utilizzo di chiavi pubbliche a 1024 bit.

4.1.2 I moduli di prelude

Passiamo ora a considerare i vari moduli che compongono di cui è composto il sistema Prelude IDS.

4.1.2.1 Librelude

La libreria Libprelude è stata generata per facilitare lo sviluppo dei file che compongono i sensori, fornendo loro caratteristiche comuni: questa libreria ha infatti lo scopo di interfacciare i sensori con il manager (i manager) di Prelude IDS. In figura 4.7 viene mostrato lo schema dell'archiettura interna di Libprelude.

La libreria Libprelude si prence carico di molte operazioni:

- *Gestione della connessione con il manager*: oltre alle comunicazione per lo scambio di messaggi IDMEF, la libreria effettua periodiche connessioni di controllo

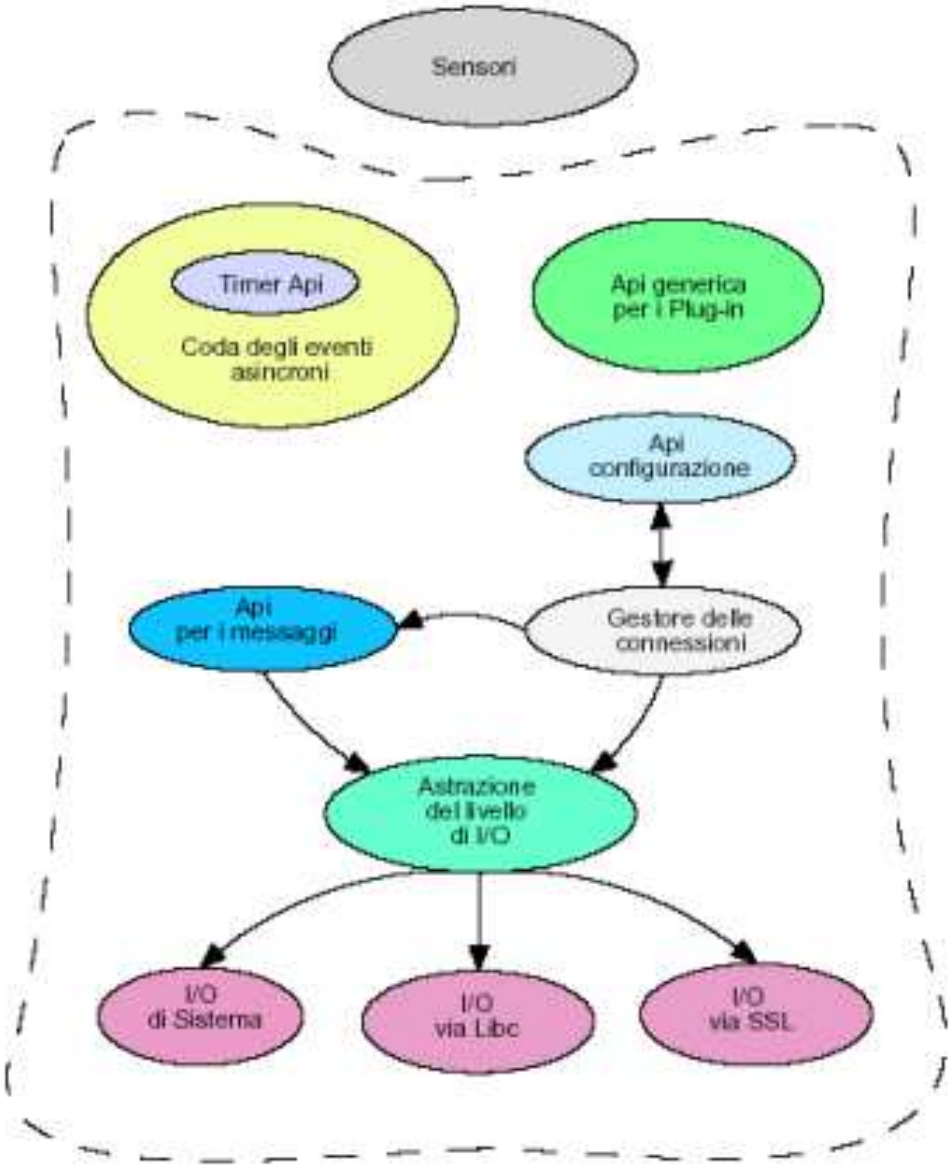

Figura 4.7: Architettura di Libprelude

con il manager; se quest'ultimo non risponde, gli allarmi vengono automaticamente salvati in un file temporaneo per poi essere considerati una volta che venga ristabilita la connessione con il manager.

- *Livello di comunicazione trasparente*: ogni qual volta vi sia un collegamento con il manager, Libprelude sceglie autonomamente il protocollo più adatto (clear-text o SSL). La libreria gestisce comunque la connessione con il manager in maniera totalmente astratta.

- *Gestione asincrona degli eventi*: tale modulo permette ai sensori di gestire in maniera asincrona gli eventi registrati; in questo modo un messaggio connesso con un evento asincrono viene trasmesso senza creare un'ostruzione all'interno della rete.

- *interfaccia per l'utilizzo di timer sincroni ed asincroni*: viene fornita la possibilità di generare timer sincroni ed asincroni. Quando un timer si esaurisce viene chiamata una funzione che ha come parametri i dati associati al timer; tale funzione ha lo scopo di riprogrammare il timer stesso.

- *Interfaccia per i file di configurazione*: fornisce un'astrazione delle opzioni da linea di comando, dei file di configurazione e dei comandi di configurazione provenienti dal manager.

- *Interfaccia di gestione dei plug-in generici*: viene data la possibilità di generare plug-in personali, utilizzando **libtdl** per la portabilità del codice sorgente.

I sensori collegati a libprelude usano i propri file di configurazione o ereditano le opzioni comuni definite nel file di configurazione del sensore globale localizzato in:

```
/usr/local/etc/prelude-sensors/sensors-default.conf
```

Tipicamente questo file di configurazione viene utilizzato per definire dei manager di ripristino in casi di fallimento. Consideriamo il seguente esempio:

```
manager-addr = 10.0.1.250:5554 || 192.168.1.251 && 192.168.1.252;
```

Questa istruzione comunica al sensore di spedire gli allarmi al manager avente indirizzo IP 10.0.1.250, sulla porta 5554; qualora tale manager risulti irraggiungibile, il sensore dovrà spedire i messaggi sia al manager con indirizzo IP 192.168.1.251, sia la manager avente indirizzo IP 192.168.1.252.

4.1.2.2 Libsafe

Libsafe è una libreria precaricabile che ha il compito di proteggere un programma contro lo sfruttamento delle vlunerabilità di tipo *buffer overflow*. Allo stesso tempo, Libsafe è in grado di verificare chiamate a funzioni insicure che non verificano la dimensione dei buffers.

Libsafe sostituisce l'esecuzione di queste funzioni e si assicura che non vi sia nessun buffer overflow che possa bloccare lo stack; essa, inoltre, intercetta la chiamata alla funzione originale e modifica il percorso di esecuizione per sostituirlo con il proprio.

Se viene rilevato un buffer overflow, Libsafe arresta l'esecuzione del programma (nel caso il programma in questione fosse un processo figlio, viene arestato solo quest'ultimo e non anche il processo padre).

Se Libsafe è stata compilata su un sistema in cui è presente anche Libprelude, Libsafe si trasforma automaticamente in un sensore per Prelude IDS. Un comportamento errato rilevato da libsafe durante l'esecuzione condurrà alla generazione di un allarme che verrà trasmesso al manager centrale di Prelude IDS. L'uso combinato di Libsafe e Libprelude permette la manager di ottenere informazioni molto precise sull'attacco rilevato.

4.1.2.3 Prelude-NIDS

Si tratta di un sensore di tipo NIDS (vedi capitolo 4); esso quindi ha la funzione di sniffing e di analisi in tempo reale dei pacchetti che transitano all'interno della rete. Prima di entrare nel dettaglio, presentiamo i figura 4.8 la struttura interna di questo modulo.

Per poter catturare i pacchetti di rete, Prelude-NIDS utilizza una versione modificata di PCAP [LIBPCAP].

Quando un pacchetto viene ricevuto, il sensore ne decodifica l'header e lo immagazzina in una struttura di dati interna. Le varie prove per determinare se il pacchetto in esame può essere considerato valido o meno sono effettuate in questa fase. Vengono effettuate rilevazioni alla ricerca di dati creati ad hoc per rendere il sistema instabile; altre prove sono effettuate sui flags di TCP e di IP. Se viene rilevata un'anomalia, il pacchetto viene rifiutato per impedire il fallimento di ulteriori analisi, come la deframmentazione dell'IP od il riassemblaggio del flusso di TCP.

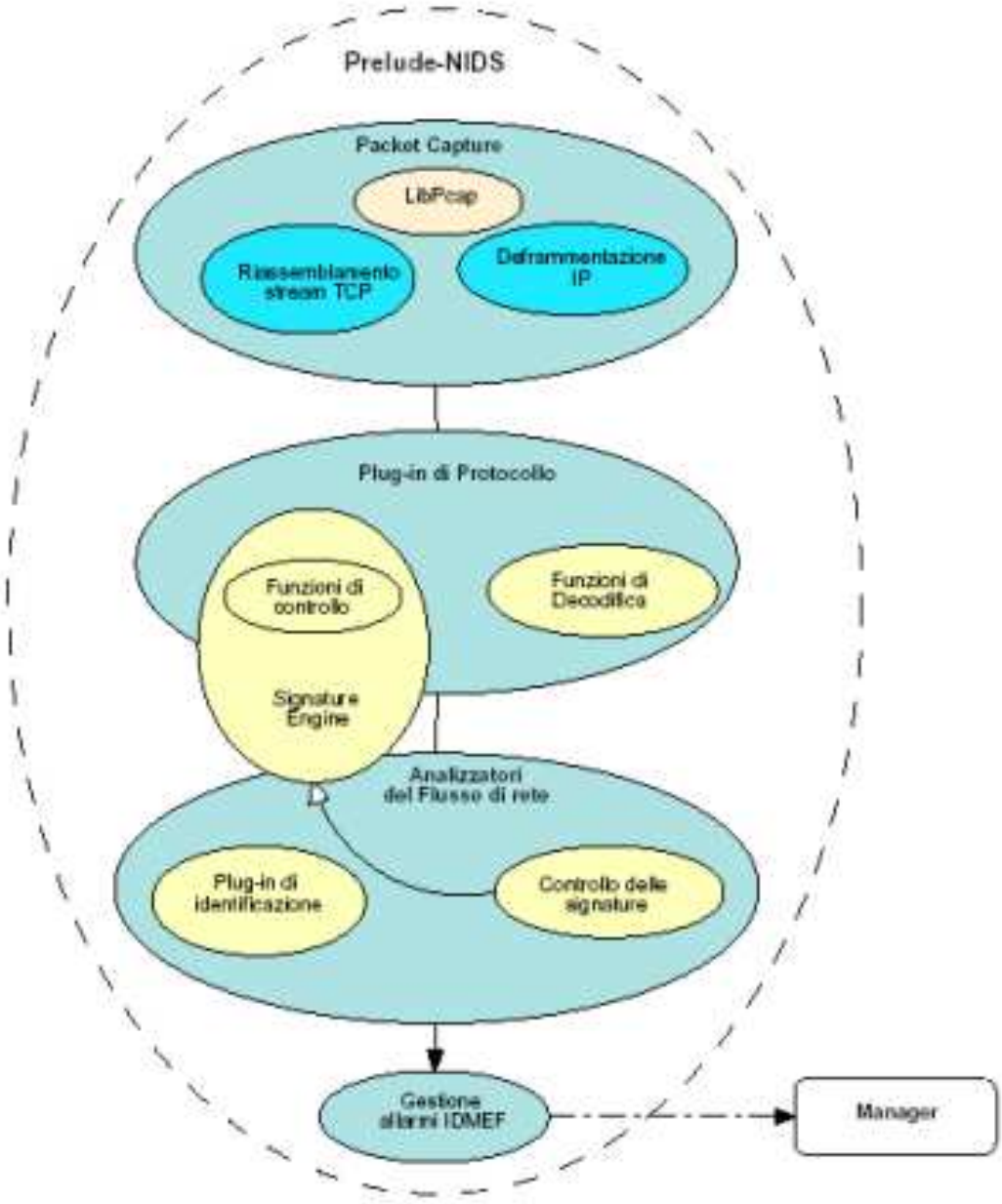

Figura 4.8: Componenti principali di Prelude-NIDS.

Se in queste prime fasi di controllo di validità del pacchetto non vengono rilevati problemi, si passa alle fasi di deframmentazione dell'IP e di riassemblaggio del flusso di TCP. Se queste caratteristiche sono attive e se il pacchetto ricevuto fa parte di una connessione TCP, allora l'analaisi relativa a quella connessione verrà ritardata sino al completamento del riassemblaggio.

Illustriamo ora in dettaglio i passi precedentemente descritti. Il file di configurazione di Prelude-NIDS è:

```
/usr/loca/etc/prelude-nids/prelude-nids.conf
```

Questo file contiene numerose opzioni raggruppate in diverse sezioni seguendo il formato **.ini**.

In questa fase i dati del pacchetto sono trasmessi al plug-in di protocollo; questi plug-in possono decodificare protocolli di livello più alto, come descritto qui di seguito:

- Normalizzatori di Dati
  - *Plug-in di decodifica HTTP*: questo plug-in cerca richieste HTTPnm nei pacchetti TCP e quando una di queste richieste viene trovata normalizza tutti i caratteri di escape generati per eludere l'identificazione di un attacco. Può inoltre rilevare un certo numero di attacchi, sequenze UTF-8 invalide o sequenze UTF-8 create per nascondere caratteri ASCII. Questo plug-in esporta le opzioni relative alla specificazione del numero di pagina del codice da utilizzare ed il file che contiene la tabella di conversione Unicode-ASCII.
  - *Caratteri di escape codificati in FTP e TELNET*: questo plug-in decodifica le pozioni TELNET o FTP nei pacchetti di dati. Esso prova a normalizzare queste opzioni per assicurarsi che non si stia cercando di interferire con gli algoritmi di string matching.

- L'header di decodifica delle firme digitali
  - *Plug-in di decodifica RPC*: questo plug-in decodifica i pacchetti UTP ed il TCP che contengono un header RPC. Se lo trova, esso analizza la firma attraverso il test:

```
rpc: function, version, program
```

Terminata questa fase si può dare inizio alla vera analisi e per fare questo il pacchetto viene trasferito al plug-in di rilevamento.

Esistono altri plug-in:

- *Plug-in Snort Rules*: si tratta di un plug-in compatibile con le firme di Snort [SNORT]. E' utilizzato da un motore generico di firme di Prelude; fornisce un parser e delle funzioni che permettono di testare la validità di un pacchetto utilizzando le regole di Snort.

- Scan detection plug-in: genera un allarme se rileva che in un certo lasso di tempo si ha un alto numero di connessioni su porte differenti. Per essere meno vulnerabile ai falsi allarmi, questo plug-in differenzia le porte privilegiate da quelle non, settando limiti differenti. In effetti le porte più interessanti sono quelle comprese da 0 a 1024.

- *Plug-in di rilevazione del codice shell polimorfico*: prima dello sviluppo di questo plug-in, negli attacchid i buffer overflow veniva utilizzata molto frequentemente l'istruzione *NOP (0x90) on IA-32)*. Era molto semplice per un IDS contare il numero di NOP nei dati e mandare un allarme se il numero di NOP raggiungeva un dato limite. Con il passare del tempo il codice shell di tipo polimorfico è stato sempre più utilizzato e la tecnica di rilevamento precedente è diventata inevitabilmente obsoleta: il polymorfic code shell infatti contiene istruzioni differenti da *NOP*, ma che generano gli stessi effetti, oppure sono in gradi di generare autonomamente il proprio codice rendendo di fatto impossibile avere firme affidabili. Sono questi i motivi che hanno spinto alla creazione di questo plug-in che è in grado di trovare un cammino di codice senza effetti. Se il numero di istruzioni eccede il limite viene generato un allarme.

- *Plug-in ArpSpoof*: il plug-in ArpSpoof controlla la coerenza dei messaggi ARP e degli headers Ethernet. Viene generato un allarme se:

  1. Una richiesta di ARP è mandata ad un indirizzo UNICAST (una richiesta valida ARP è mandata all'indirizzo BROADCAST) attraverso l'opzione *directed*. Una richiesta UNICAST ARP denota spesso un attacco.

2. L'indirizzo di origine ETHERNET è differente da quello incluso nel messaggio ARP.
3. L'indirizzo di destinazione ETHERNET è differente da quello incluso nel messaggio ARP.
4. Esso permette di specificare l'indirizzo IP/MAC mappando attraverso *arpwatch* e rileva ogni messaggio ARP in conflitto con il *database arpwatch*.

Il pacchetto è mandato con un meccanismo di firme. L'architettura logica del motore delle firme è interna a Prelude. Gli altri componenti, come il parser della firma o il test specifico di una firma, sono implementati dai plug-in.

- *Deframmentazione degli IP*: la frammentazione degli IP è usata quando sono spediti ad un'altra macchina più dati del possibile. Il limite è settato dalla MTU (Unità Massima di Trasferimento). Se la comunicazione intreccia diversi punti, la minimo MTU tra punti finali fissa la massima dimensione dei pacchetti durante la trasmissione. Un hacker può usare la frammentazione dell'IP per fuggire dall'IDS, disturbando l'algoritmo di string matching. Per reagire a questo effetto, Prelude ha uno stack di deframmentazione dell'IP copiato da Linux.

- *Riassemblamento dei flussi TCP*: da un po' di tempo, Prelude è in grado di riassemblare i flussi TCP (opzione *tcp-reasm*), che protegge se stesso da diversi tipi di attacco:

  – Attacchi senza stato

    Esistono metodi per inquinare i log generati da applicazioni come Prelude-NIDS. Applicazioni come IDSWakeUp, Stick e Snort mandano pacchetti creati per generare falsi allarmi e nascondere quelli veri all'analizzatore. Queste applicazioni non attaccano veramente la macchina, ma creando un pacchetto conforme ad una regola dell'engine signature (ma che non è parte di una connessione), permettono ad un attacco di inquinare i log con la conseguente impossibilità da parte dell'analizzatore di comprenderli e risolvere la situazione. In casi di diverse migliaia di allarmi generati, l'analizzatore avrà molte difficoltà a trovare gli eventi veramente importanti.

4.1.2.4 Prelude Manager

Prelude Manager è un server che gestisce i diversi collegamenti con i sensori centralizzando l'analisi dei loro dati. Uno schema generale di questo server è rappresentato in figura 4.9.

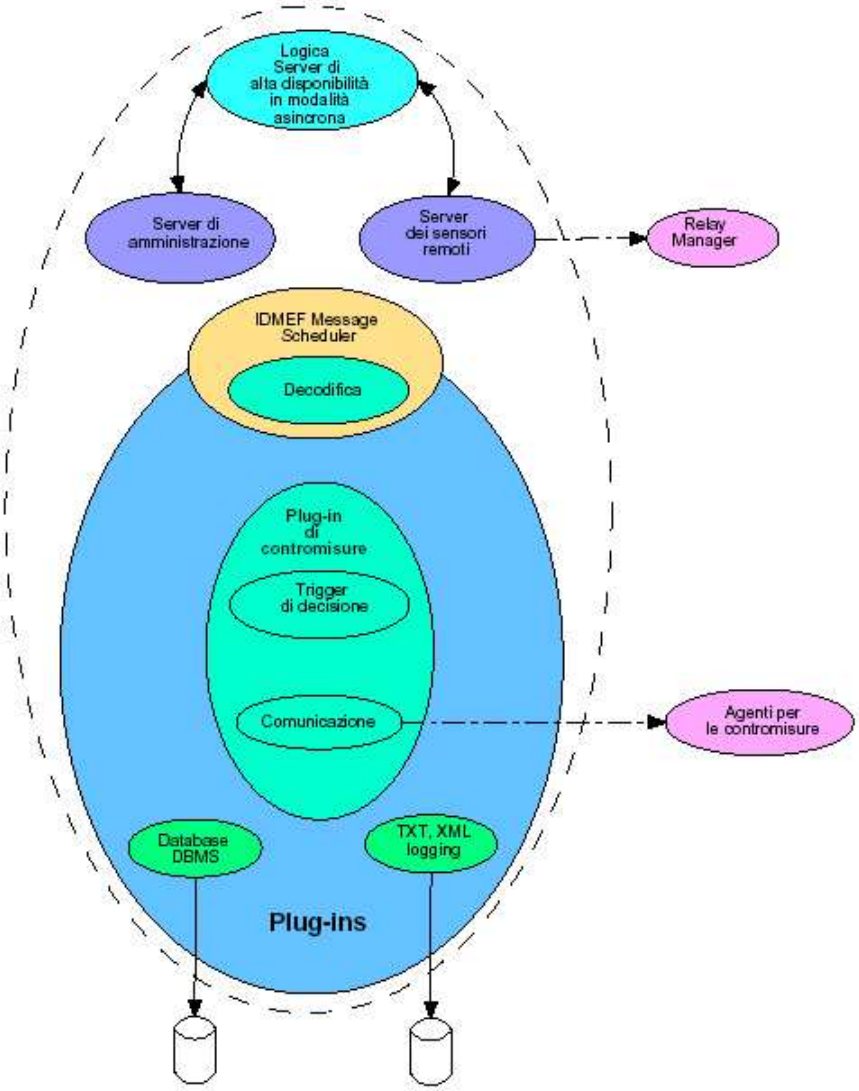

Figura 4.9: Componenti fondamentali del manager

Quando un sensore rileva un'attività anomala trasmette un allarme. Il server thread del manager rileva la presenza di dati su uno dei suoi pool, quindi legge il

messaggio (in senso non bloccante) che a sua volta è trasmesso ad uno scheduler.

Lo scheduler dei messaggi allora decide, in base ai vincoli di memoria ed alla priorità del messaggio, se questo deve essere conservato nella memoria per subire un'elaborazione veloce od essere scritto provvisoriamente sul disco per essere processato successivamente.

Quando un messaggio può essere processato, viene convertito dal suo stato binario alla versione IDMEF; tale conversione è di tipo non distruttivo, ciò signigica che viene mantenuta la versione binaria originale del messaggio.

Durante la fase di lettura del messaggio può renderesi necessario l'utilizzo di un particolare plug-in di decodifica del formato IDMEF in quanto quest'ultimo non è particolarmente adatto a rappresentare alcuni dati forniti dai sensori. Per questo motivo questo tipo di plug-in convertono i dati grezzi nel formato IDMEF nel modo più opportuno.

Supponiamo, per esempio, che il sensore Prelude-Nids voglia mostrare all'utente il pacchetto che ha generato un determinato allarme. Il formato IDMEF non è però predisposto a rappresentare in modo chiaro la struttura del pacchetto. Per questo motivo, un plug-in deve decodificare il pacchetto e trasformarlo in dati addizionali, leggibili nell'allarme in formato IDMEF.

Il messaggio generato da IDMEF viene trasmesso al modulo "contromisure" di Prelude, costituito da un insieme di plug-in. Un plug-in triggered decide, secondo la configurazione e l'allarme, se una contromisura è necessaria.

In questo caso, il triggering plug-in manda una richiesta al plug-in di comunicazaione, il quale trasmette il messaggio ad un agente distribuito che può rispondere.

Il messaggio è allora spedito al plug-in di segnalazione ed al database il quale si prende cura di modificarlo nel formato più consono alla sua memorizzazione. Tale fromato è scritto nelle specifiche del plug-in e permette d'immagazzinare il messaggio di allarmein un database, in un file, o in altre strutture dati.

I plug-in di segnalazione e i database disponibili sono:

- *MySQL Plug-in*
- *PostgreSQL Plug-in*
- *Text Teport Plug-in*

4.1.2.5 Console di amministrazione

Il Prelude manager ha una console di amministrazione. Infatti, ogni sensore manda al manager una lista di opzioni, la quale può essere settata in remoto. Il manager trasforma la lista di opzioni in formato XML e questa è data ai client i quali la chiedono al server.

4.1.2.6 Relè Manager

I manager possono anche funzionare come relè. Questo è molto utile quando una rete è divisa in sotto-reti, l'agente delle contromisure è installato in una di esse ed il logging è centralizzato.

Per abilitare la trasmissione, si usa l'opzione *relè-manager* nel file di configurazione o sulla linea di comando. Per esempio, per trasmettere gli allarmi a 192.168.1.27 e deviarli, in caso di fallimento, a 192.168.2.51 e 192.168.2.53, basta settare:

```
relay-manager = 192.168.1.27 || 192.168.2.51 && 192.168.2.52;
```

La configurazione del prelude-manager è scritta nel file:

```
the/usr/local/etc/prelude-manager/prelude-manager.conf
```

4.1.2.7 Prelude-LML

Prelude Log Monitoring Lackey è una parte del progetto che tratta con gli host basati sugli aspetti di intrusion detection (vedi figura 4.10).

Da un lato, centralizza i log che provengono da piattaforme differenti. Dall'altro lato, analizza questi log in sequenza, per scoprire quale informazioni potrebbero essere importanti per quanto riguarda la sicurezza. Le differenti piattaforme supportate sono le macchine che usano il protocollo di **BSD syslog**:

- Sistemi Unix
- Switches e routers
- Firewalls
- Stampanti
- Altri sistemi che possono trasformare i loro log nel formato syslog (come Windows NT/2K/XP con i tools come Ntsyslog), ...

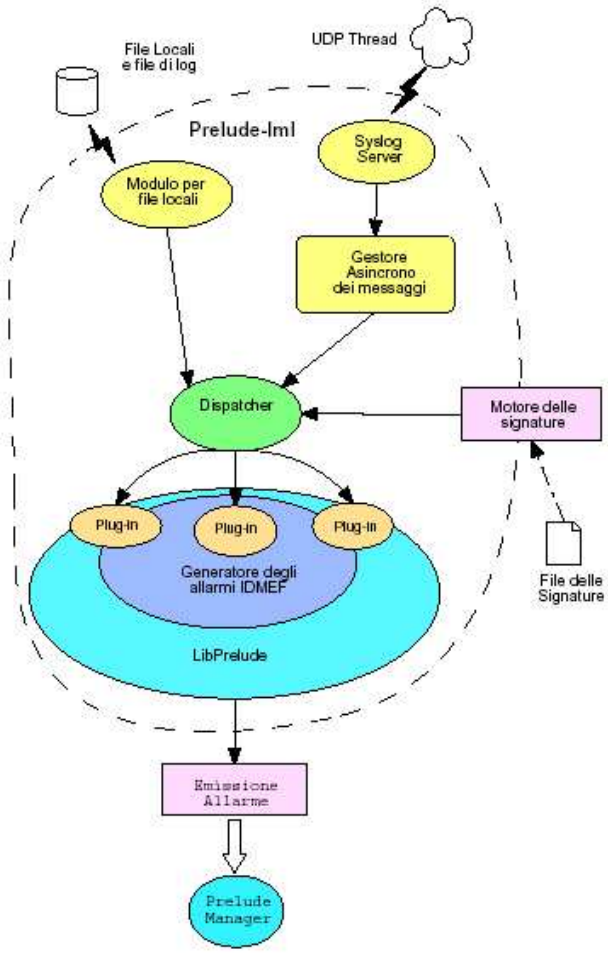

Figura 4.10: Componenti fondamentali di Prelude-LML

E' quindi possibile centralizzare tutte le informazioni derivanti dall'attività di logging installando un unico Prelude-LML all'interno di una rete (al fine di monitorare i dati) e configurando tutte le altre macchine per trasmettere i loro log al daemon di syslog simulato da LML. Questa caratteristica dell'unificazione dei log nel formato syslog è attivata nel file di configurazione di Prelude-LML (che si trova in `/usr/local/etc/prelude-lml/prelude-lml.conf`) nel seguente modo:

```
[Udp-Srvr]
port = 514            # Listening port (514 by default)
addr = 192.168.1.99   # Listening address (0.0.0.0 by default)
```

o direttamente dalla linea di comando:

```
/usr/local/bin/prelude-lml --udp-srvr --addr 192.168.1.99 --port 514
```

E' anche possibile installare Prelude-LML localmente su una singola macchina Unix, utilizzandolo non server syslog in ascolto degli allarmi da rete, ma come un analizzatore di log locale. Infatti, Prelude-LML è in grado di operare in entrambe le modalità grazie a un set di plug-in che ricercano attività rilevanti per l'analisi della sicurezza. Questi plug-in sono chiamati usando la libreria *PCRE* e la loro configurazione è settata di default nel file:

```
/usr/local/etc/prelude-lml/plugins.rules
```

Questo file indica quali plug-in devono essere chiamati quando una o altre espressioni regolari trovano una linea di allarme.

Poiché Prelude-LML usa la libreria libprelude (vedi sezione 4.1.2.1), è possibile riportare tutti gli allarmi tramite connessioni SSL, dove l'integrità e la riservatezza dei messaggi può essere assicurata, così come l'autenticità fra i sensori ed il responsabile con il quale stanno comunicando.

Quando i log sono letti, analizzati e trasformati, se necessario, nella formato IDMEF implementato nella libreria libprelude, vengono spediti attraverso connessioni SSL ad un manager.

4.1.2.8 Prelude-PHP-Frontend

Prelude-PHP-Frontend è la parte dell'interfaccia web della gestione del progetto di Prelude. In figura 4.11 è rappresentata una tipica schermata ottenuta accedendo al Frontend tramite un browser internet.

Costituito da molti script PHP, è basato su Apache e sulle librerie Adodb e PhPlot.

Dal punto di vista architetturale, il manager scrive gli allarmi del formato IDMEF nel suo database (funzione *(1)saves* dello schema in figura 4.12). Questo obbiettivo può essere raggiunto usando richieste standard SQL sia localmente che su sistema remoto.

Abbiamo quindi una relazione funzionale del tipo mostrato nello schema 4.12.

Il frontend limita l'accesso all'Interfaccia Administation Web usando degli account. Usando gli account si permette di definire parecchi profili di amministrazione. E' molto importante nelle reti di grandi dimensioni, con architetture complesse,

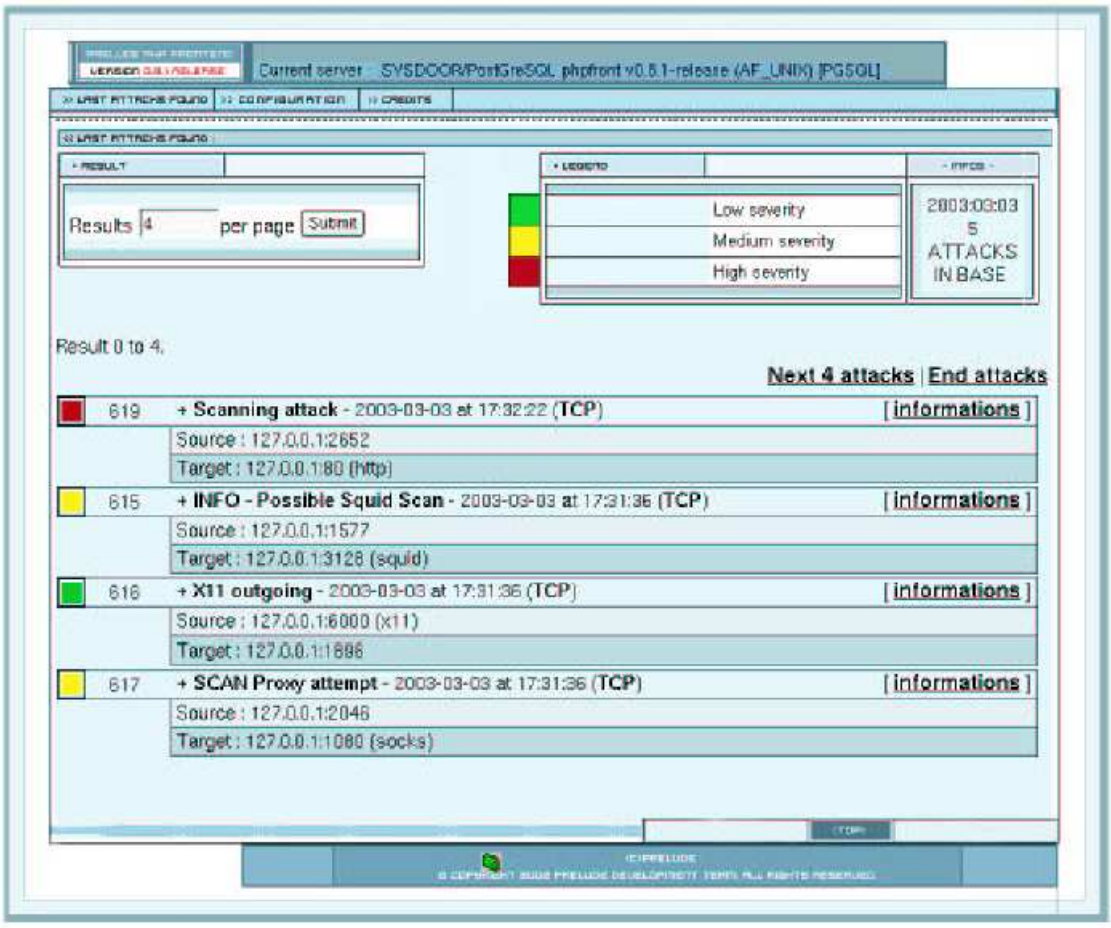

Figura 4.11: La finestra di visualizzazione degli allarmi nel frontend di Prelude

poter aver profili sicuri degli amministratori. Inoltre, una volta collegati al Prelude-PHP-Frontend, si possono definire le disposizioni differenti di gestione che dipendono da che cosa si desidera osservare. L'interfaccia web permette che si configuri le viste di cui si è interessati.

4.2 Messaggi IDMEF

L'Intrusion Detection Message Exchange Format [IDMEF] è stato concepito allo scopo di fornire un formato standard che i sistemi di Intrusion Detection possono usare per riportare allarmi riguardanti eventi che essi ritengono sospetti.

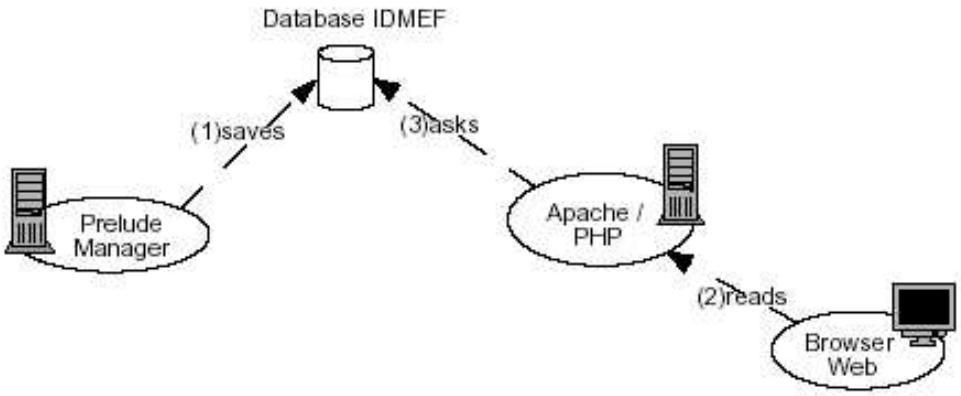

Figura 4.12: Schema di funzionamento di Prelude Frontend

Questa formattazione dei dati non viene utilizzata unicamente per creare messaggi scambiati fra le varie compnenti del sistema IDS distribuito: essa è infatti utilizzata anche per altri scopi.

Un singolo database può immagazzinare i risultati di una varietà di intrusioni in modo tale che questi vengano poi analizzati per poter avere una fotografia realista del sistema di cui si dispone, correlata cioè dei punti in cui è particolarmente debole e vulnerabile, per poter così realizzare eventuali contromisure e potenziamenti difensivi del sistema stesso;

Un sistema di correlazione degli eventi può accettare allarmi da una grande varietà di intrusion detection software e in seguito potrà essere capace di eseguire i più sofisticati calcoli individuando correlazioni incrociate nel sistema analizzato, e questo porterà ancora alla scoperta di contromisure maggiormente esaustive per la difesa del proprio sistema;

Un'interfaccia utente può mostrare gli allarmi di una varietà di intrusioni monitorando in tempo reale o meno il sistema sotto controllo;

Un formato di scambio di dati comune potrà facilitare la comunicazione tra più organizzazioni (utenti, rivenditori, enti legali) tra loro autonome, cosa abbastanza improbabile e imprecisa se i dati scambiati non appartengono ad uno standard comune a tutte.

4.2.1 IDMEF Data Model

L'IDMEF Data Model è una rappresentazione ad oggetti dei dati riguardante gli allarmi spediti dai sensori del sistema agli intrusion detection manager. Il Data Model è stato disegnato per rappresentare gli allarmi in modo non ambiguo e in maniera tale da poter descrivere le relazioni tra gli allarmi semplici e quelli complessi. Con il termine "rappresentazione non ambigua" si intende che, se da un lato si permette ai sensori di produrre più o meno informazioni all'interno degli allarmi da essi generati, dall'altro non si permette loro di produrre informazioni contraddittorie in due allarmi riferiti allo stesso evento.

Questo modello presenta diversi problemi a causa della rappresentabilità dei dati inerenti agli allarmi:

- le informazioni sugli allarmi sono particolarmente eterogenee. Alcuni allarmi sono definiti con pochissime informazioni (come per esempio origine, destinazione, nome e ora in cui tale evento si è verificato), altri invece includono maggiori informazioni (come elenchi di porte in cui la macchina è stata attaccata, i processi coinvolti, le informazioni sull'utente, etc). Il modello di dati che rappresenta queste informazioni deve quindi essere flessibile per adattarsi a questa eterenogeneità.

- Un modello orientato agli oggetti deve naturalmente essere estensibile tramite aggregazioni e sottoclassi. Se un'implementazione del Data Model viene estesa con nuove classi (tramite aggregazione o ereditarietà), un'implementazione che non conosce queste estensioni rimarrà comunque in grado di comprendere il sottoinsieme di informazioni che è definito dal Data Model originario. L'ereditarietà e l'aggregazione forniscono estensibilità preservando al contempo la consistenza del modello originale.

- Gli ambienti di intrusion detection sono molteplici. Alcuni sensori rilevano attacchi analizzando il traffico di rete, altri usano i log dei sistemi operativi. Per questi motivi, due o più allarmi, mandati da sensori con differenti origini di informazioni, ma relativi al medesimo attacco, non conterranno gli stessi dati.

  Il data model definisce classi di supporto che permettono di annullare le differenze nelle sorgenti dei dati tra i sensori. Per esempio, la nozione di sorgente e

di destinazione per l'allarme è rappresentata da una combinazione delle classi *Node*, *Process*, *Service* e *User*.

- Le capacità dei sensori sono differenti: si possono installare sensori "leggero", che forniscono cioè poche informazioni all'interno degli allarmi, oppure sensori più complessi che avranno un grande impatto sul sistema ospite (in termini di risorse di sistema necessarie al loro funzionamento), ma in grado di fornire informazioni più dettagliate sull'allarme generato. Il data model deve favorire la traduzione verso formati diversi, utilizzati da strumenti di analisi di dati, al fine di permettere ulteriori strudi delle informazioni relative agli allarmi.

- Il data model definisce estensioni di uno schema base che permette di rappresentare allarmi sia semplici che complessi. Le estensioni sono implementate per mezzo della definizione di sottoclassi o attraverso l'associazione di nuove classi.

- Gli ambienti operativi sono molteplici. Dipendendo dal tipo di rete e dal sistema operativo utilizzato. Gli attacchi saranno osservati e riportati con diverse caratteristiche ed il data model deve essere in grado di considerare queste differenze.

4.2.2 Implementazione XML di IDMEF

Originariamente, furono proposte all'IDWG (Intrusion Detection Working Group) [IDWG] due possibili implementazioni dell'IDMEF: una basata su di una Structure of Management Information (SMI) per descrivere un SNMP MIB, e l'altra basata su di un Document Type Definition (DTD) per descrivere documenti XML.

Queste proposte vennero analizzate negli incontri dell'IDWG nel settembre 1999 e nel febbraio 2000, e venne deciso che la soluzione XML rispondeva in maniera più adeguata alle richieste dell'IDWG.

XML è un metalinguaggio che permette ad un'applicazione di definire il proprio mark-up. Per questo le applicazioni basate su 'XML sono state usate o sviluppate per una vasta varietà di scopi. Questa sua flessibilità lo innalza al ruolo di linguaggio ideale per l'implementazione del formato IDMEF, dal momento che:

- l'XML permette la creazione di un linguaggio personalizzato, sviluppato al solo fine di descrivere allarmi di intrusion detection. Inoltre definisce uno anche standard per l'estensione del linguaggio stesso.

- Gli strumenti software per eseguire documenti XML sono largamente disponibili, sia in formato commerciale che open source. Numerosi strumenti e APIs per il parsing e la validazione di documenti XML sono disponibili in vari linguaggi, inclusi Java, C, C++, Tcl, Perl, Python, e GNU Emacs Lisp.
- XML incontra i requisiti IDMEF elencati nella sezione 4.2.1. Lo standard XML richiede supporti sia per la codifica UTF-8 che la UTF-16 di ISO/IEC 10646 e Unicode, rendendo così tutte le applicazioni XML (e quindi anche tutte le applicazioni che utilizzano il formato IDMEF) compatibili con queste codifiche dei carattere comuni.
- I progetti sviluppati in XML saranno provvisti di estensioni orientate agli oggetti, supportati da database, e altri caratteristiche utili.
- XML è gratuito, senza licenza e senza royalities.

4.2.3 Struttura dei messaggi

In questa sezione, sono spiegati in dettaglio i singoli componenti dell'IDMEF data model. I diagrammi UML del modello sono forniti per mostrare come ciascun componente è relazionato con gli altri. La relazione tra i principali componenti del data model è mostrata in figura 4.13.

Per tutti i messaggi IDMEF, la classe a livello più alto è la IDMEF-Message: ogni tipo di messaggio è sottoclasse di questa. Sono quindi definiti due tipi di messaggi: gli Alerts e gli Heartbeats. Le altre sotto classi servono per fornire maggiori informazioni su questi messaggi.

E' importante notare come il data model non specifichi come gli allarmi debbano essere classificati o identificati. Per esempio, un port scan può essere identificato da un sensore come singolo attacco contro molteplici obbiettivi, mentre un altro sensore può identificarlo come attacchi multipli da una singola sorgente.

Comunque, una volta che un sensore ha determinato il tipo di allarme da mandare, il data model impone come l'allarme deve essere formattato.

4.2.3.1 La classe IDMEF-Message

Tutti i messaggi IDMEF sono istanziati dalla classe IDMEF-Message. Essa risiede al livello più alto del data model e contempla due sole sottoclassi: Alert e Heartbeat.

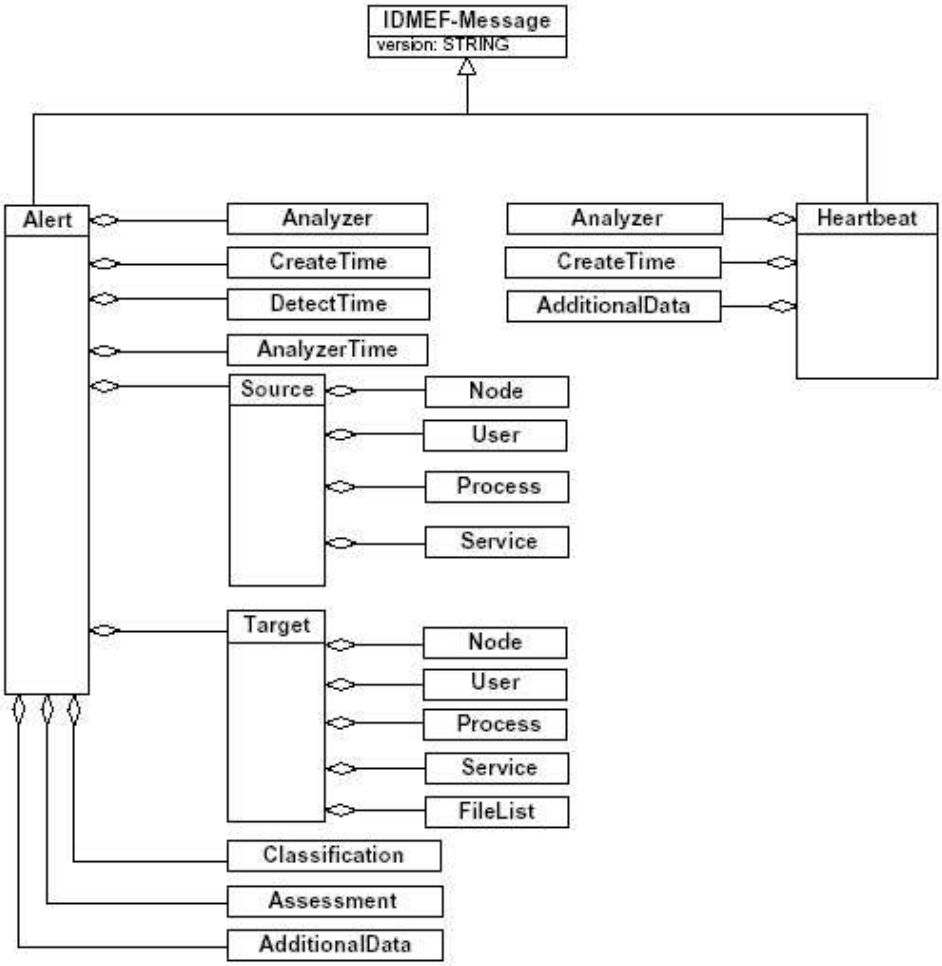

Figura 4.13: Il data model

version è l'unico attributo di IDMEF-Message e rappresenta la versione di IDMEF utilizzata.

4.2.3.2 La classe Alert

Generalmente, ogni volta che un sensore rileva un evento per il quale è configurato, manda un messaggio di allarme ai suoi manager. Dipendendo dal sensore, un messaggio di allarme può corrispondere ad un singolo evento rilevato oppure ad una molteplicità di eventi rivelati. Ogni allarme viene mandato asincronicamente come risposta ad un evento esterno.

Un messaggio di allarme è composto da diverse classi aggregate, come mostrato in figura 4.14. Ogni classe è descritta dettagliatamente nei paragrafi seguenti.

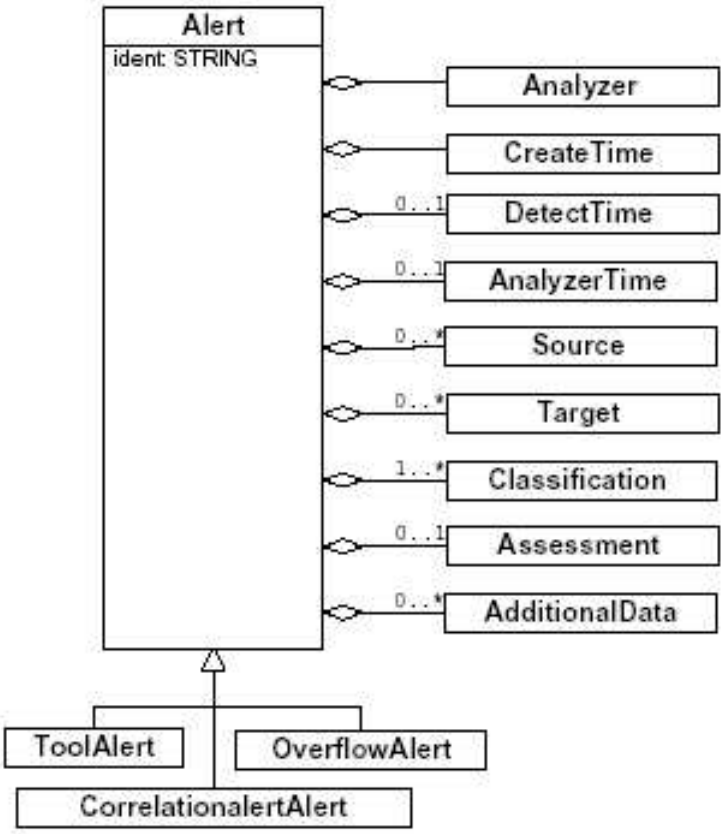

Figura 4.14: La classe Alert

Le classi aggregate che costituiscono l'allarme sono:

Analizer Identifica le informazioni per i sensori che originano l'allarme.

CreateTime L'ora in cui l'allarme è stato creato. Dei tre tipi di informazioni temporali è l'unica obbligatoria.

DetectTime Identifica l'ora in cui il primo evento che ha generato l'allarme è avvenuto. In alcune circostanze, potrebbe non essere lo stesso valore del **CreateTime**.

AnalyzerTime L'ora corrente sul sensore.

Source E' la sorgente (o le sorgenti) dell'evento.

Target E' il bersaglio (o i bersagli) dell'evento.

Classification E' il "nome" dell'allarme, o altre informazioni che permettono al manager di determinare cos'è.

Assessment Informazioni relative all'impatto dell'evento, le azioni prese dal sensore per rispondere ad esso e il grado di sicurezza che il sensore stesso dà a questa valutazione.

AdditionalData Informazioni inerenti il sensore che non sono adatte a essere rappresentate nel data model. Queste possono consistere in singoli frammenti di dato come a notevoli quantità di dati provenienti dalle estensioni dell'IDMEF.

ident è l'unico attributo della classe Alert.

4.2.3.3 La classe ToolAlert

Tale classe porta informazioni addizionali relative all'uso di tools di attacco o programmi maligni come un Trojan horse e può essere usata da un sensore quando è in grado di identificare questi tools. Esso raggruppa uno o più dei precedenti allarmi inviati insieme per evidenziare come essi erano stati creati utilizzando quel tool specifico.

Questa classe è composta da tre classi aggregate, come si vede in figura 4.15.

Name Esprime il motivo per il quale vari allarmi sono stati raggruppati insieme e può essere, ad esempio, il nome di un particolare tool di attacco.

Command Il comando o l'operazione che il tool ha cercato di eseguire, per esempio un BackOrifice ping.

Alertident E' la lista completa degli identificatori degli allarmi correlati a questo allarme.

4.2.3.4 La classe CorrelationAlert

Questa classe, mostrata in figura 4.16, porta informazioni aggiuntive relative alla correlazione delle informazioni relative ai vari allarmi. Il suo intento è di raggruppare allarmi mandati precedentemente per far notare come essi sono correlati.

Le classi aggregate che costituiscono la classe sono:

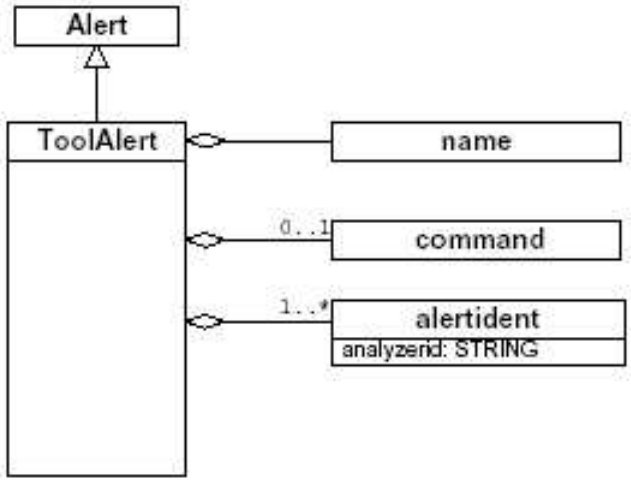

Figura 4.15: La classe ToolAlert

Name vedi sezione 4.2.3.3.

Alertident vedi sezione 4.2.3.3.

4.2.3.5 La classe OverflowAlert

Questa classe porta informazioni addizionali relative agli attacchi di tipo buffer overflow. La classe è composta da tre classi aggregate, come mostrate nella figura 4.17.

in cui:

Program Il programma che l'attacco di overflow buffer cercava di processare.

Size E' la dimensione in bytes dell'overflow.

Buffer Alcuni (o tutti) i data overflow catturati dal sensore.

4.2.3.6 La classe Heartbeat

I sensori utilizzano i messaggi di heartbeat per indicare il loro stato corrente ai manager. Gli heartbeat vengono spediti regolarmente, in genere ogni dieci minuti o ogni ora. La ricezione di un messaggio di heartbeat da un sensore indica al manager che il sensore in questione è ancora attivo; una mancanza di ricezione di un heartbeat significa che il sensore che avrebbe dovuto spedirlo non funziona

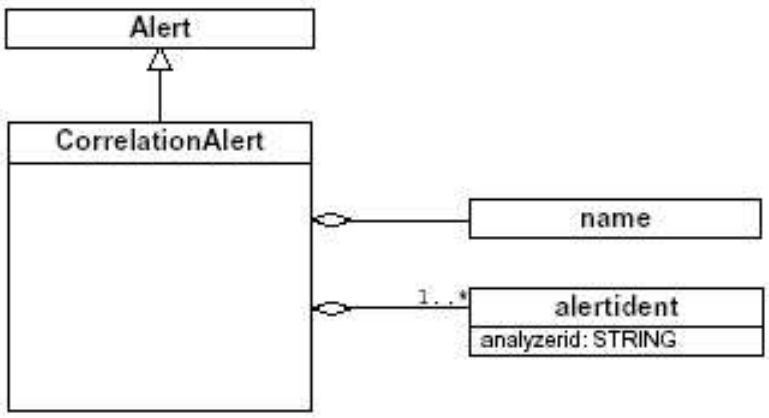

Figura 4.16: La classe CorrelationAlert

correttamente oppure che la rete di comunicazione tra il sensore e il manager è caduta.

Tutti i manager devono supportare la ricezione dei messaggi di heartbeat; comunque, l'uso di questi messaggi da parte dei sensori è opzionale. In figura 4.18 sono illustrate le classi aggregate.

Per quanto riguarda la descrizione delle classi **Analyzer**, **CreateTime**, **AnalyzerTime** e **AdditionalData** si veda la sezione 4.2.3.2.

4.2.3.7 La classe Core

La classe Core costituisce l'insieme delle classi aggregate **Analyze**r, **AdditionalData**, **Source**, **Target** e **Classification** e costituisce la parte principale della classi **Alert** e ed **Heartbeat**, come mostrato in figura 4.19.

4.2.3.8 La classe Analyzer

La classe Analyzer identifica il sensore che origina il messaggio di allarme o di heartbeat. Sebbene il data model dell'IDMEF non impedisca l'uso di sistemi intrusion detection di tipo gerarchico (dove gli allarmi attraversano i vari nodi dell'albero), esso non fornisce in ogni modo di registrare l'identità dei sensori intermedi lungo il cammino dal sensore che ha rilevato l'evento anomalo al manager che deve ricevere l'allarme.

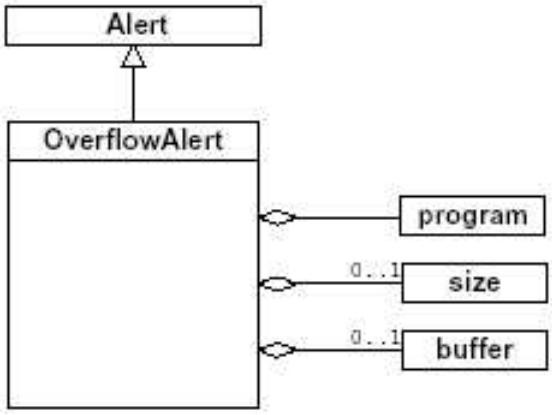

Figura 4.17: La classe OverflowAlert

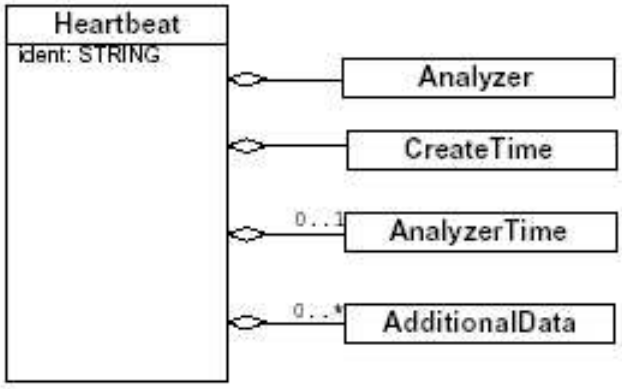

Figura 4.18: La classe Heartbeat

La classe Analyzer è composta da due classi aggregate, come mostrato in figura 4.20.

Node Informazione riguardante l'host o il dispositivo sul quale il sensore risiede (indirizzo network, nome network, ecc.).

Process Informazione sul processo sul quale è in esecuzione il sensore.

La classe Analyzer ha sette attributi:

analyzerid L'identificatore del sensore.

manufacturer Il costruttore del sensore software e/o hardware.

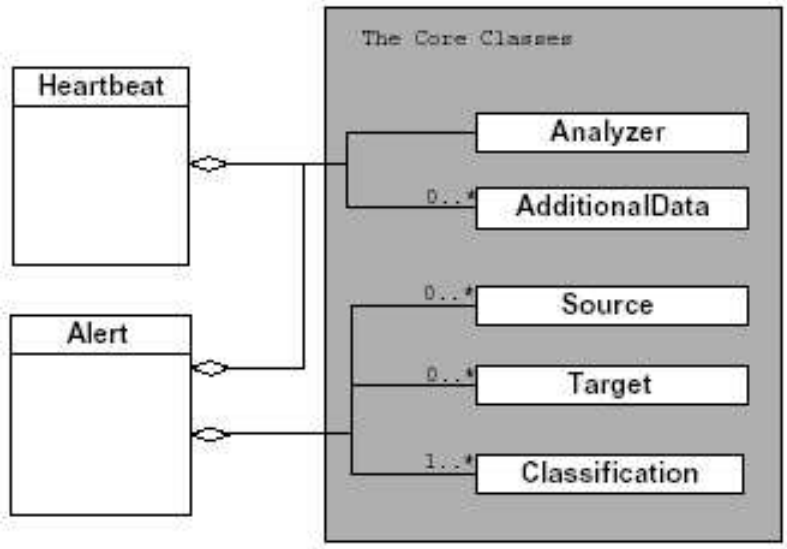

Figura 4.19: La classe Core

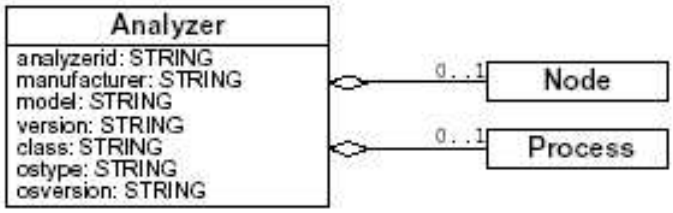

Figura 4.20: La classe Analyzer

mode Il numero/nome del modello del sensore software e/o hardware.

version Il numero della versione del sensore software e/o hardware.

class La classe del sensore software e/o hardware.

ostype Nome del sistema operativo.

osversion Versione del sistema operativo.

4.2.3.9 La classe Classification

La classe Classification fornisce il "nome" dell'allarme, o altre informazioni che permettono al manager di determinare a cosa è dovuto l'evento registrato dal sensore.

Tale classe presenta due classi aggregate, come si può vedere dalla figura 4.21.

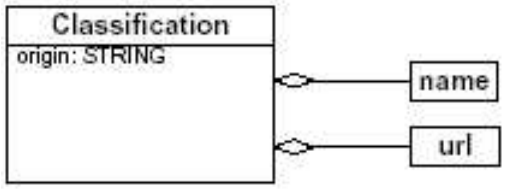

Figura 4.21: La classe Classification

Rango	Parola chiave	Descrizione
0	Unknow	Origine del nome sconosciuta
1	Bugtraqid	Identificatore del database sulla vulnerabilità (http://www.securityfocus.com/vb)
2	cve	Il Common Vulnerabilities and Exposures (CVE) (http://www.cve.mitre.org/)
3	Vendor-specific	Il nome di un rivenditore specifico (ovvero il suo URL)

Tabella 4.1: Valori dell'attributo origin nella classe classification

Name Il nome dell'allarme.

Url L'url al quale il manager può trovare informazioni addizionali sull'allarme, per esempio una descrizione più approfondita o appropriate contromisure.

La classe Classification ha un sono attributo: *origin*, che rappresenta la sorgente dal quale il nome dell'allarme viene generato. I valori permessi per questo attributo sono mostrati nella tabella 4.1 (il valore di default è `unknow`).

4.2.3.10 La classe Source

La classe Source contiene informazioni sulle possibili sorgenti degli eventi che generano l'allarme. Ogni evento può avere più di una sorgente, come, per esempio, in caso di un attacco del tipo DDOS. La classe source è composta da quattro classi aggregate (vedi figura 4.22).

Node Informazione sull'host o sul dispositivo che sembra avere causato l'evento (indirizzo del network, il suo nome, ecc,).

User Informazione sull'utente che sembra aver causato l'evento.

Process Informazione sul processo che sembra aver causato l'evento.

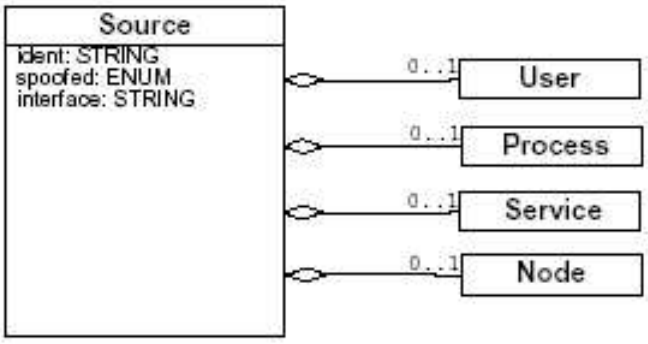

Figura 4.22: La classe Source

Rango	Parola chiave	Descrizione
0	Unknow	Non è possibile chiarire se la sorgente è reale o fittizia
1	yes	La sorgente è creduta essere fittizia
2	no	La sorgente è creduta essere reale

Tabella 4.2: Valori dell'attributo spoofed nella classe source

Service Informazione sul network service implicato nell'evento.

La classe Source ha tre attributi:

ident Identificatore unico della sorgente;

spoofed Un'indicazione sul fatto che la sorgente mostrata non è quella reale. Tale attributo può assumere uno dei valori elencati nella tabella 4.2:

interface Può essere usata da un sensore di rete con interfacce multiple per indicare su quale interfaccia è stata rilevata questa sorgente.

4.2.3.11 La classe Target

Tale classe contiene informazioni sui possibili bersagli degli eventi che hanno generato l'allarme. Un evento può avere più che un bersaglio (per esempio nel caso di port-scanning).

La classe Target (vedi figura 4.23) è composta da quattro classi aggregate la cui descrizione è già stata fatta nelle sezioni precedenti.

L'attributo *decoy* ha la funzione che aveva l'attributo spoofed nella classe Source.

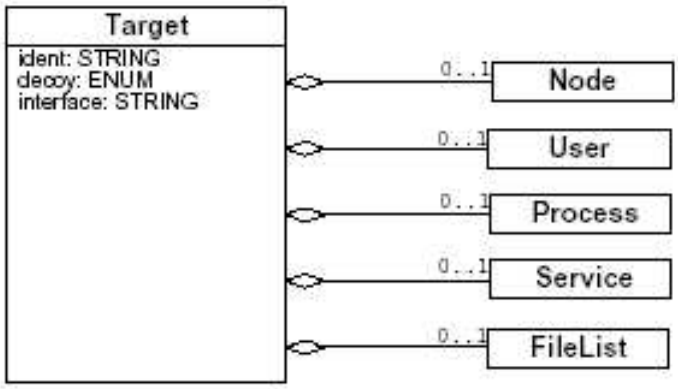

Figura 4.23: La classe Target

4.2.3.12 La classe Assessment

Tale classe è usata per fornire al sensore la capacità di accertarsi di un evento, sul suo impatto, sulle azioni prese per reagire e la fiduca che egli stesso dà alla propria stima. La sua struttura è rappresentata in figura 4.24.

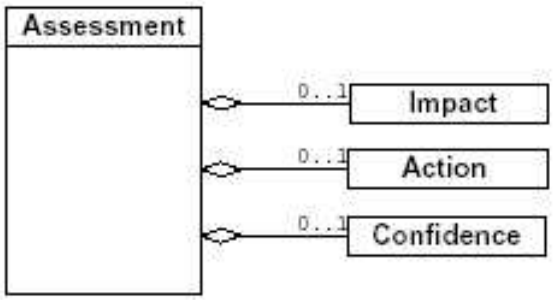

Figura 4.24: La classe Assessment

4.2.3.13 La classe AdditionalData

Questa classe è usata per fornire informazioni che non possono essere rappresentate dal data model. La classe AdditionalData può essere usata per fornire dati primitivi (interi, stringhe, ecc.) nei casi dove occorrono spedire piccole quantità di dati, oppure può essere usata per estendere il data model e il DTD per supportare la trasmissione di dati complessi.

La classe descritta presenta due attributi:

Rango	Parola chiave	Descrizone
0	boolean	L'elemento contiene un valore booleano
1	byte	L'elemento contiene un byte
2	character	L'elemento contiene un singolo carattere
3	date-time	L'elemento contiene una stringa in formato Data/Ora
4	integer	L'elemento contiene un intero
5	ntpstamp	L'elemento contiene un NTP timestamp
6	portlist	L'elemento contiene una lista di porte
7	real	L'elemento contiene un numero reale
8	string	L'elemento contiene una stringa
9	xml	L'elemento contiene un dato in formato XML

Tabella 4.3: Valori dell'attributo type nella classe additionaldata

type E' il tipo di dato incluso nell'elemento. I valori permessi da questo attributo sono mostrati in tabella 4.3. Il valore di default è string.

meaning Una stringa descrivente il significato del contenuto dell'elemento.

4.2.3.14 Le classi Time

Il data model fornisce tre classi per rappresentare l'ora. Queste classi sono classi aggregate di **Alert** e **Heartbeat**, come si può vedere dalla figura 4.25.

4.2.3.15 La classe CreateTime

La classe CreateTime è utilizzata per indicare la data e l'ora in cui l'allarme o il messaggio di heartbeat è stato creato dal sensore.

4.2.3.16 La classe DetectTime

Tale classe è usata per indicare la data e l'ora in cui gli eventi producenti un allarme vengono scoperti dal sensore. Nel caso del verificarsi di più di un evento, viene indicata l'ora del primo evento rilevato.

4.2.3.17 La classe AnalyzerTime

Questa classe è utilizzata per indicare la data e l'ora corrente sul sensore. Il suo valore deve essere riempito il più tardi possibile nel processo di trasmissione del messaggio, idealmente immediatamente prima di spedire il messaggio.

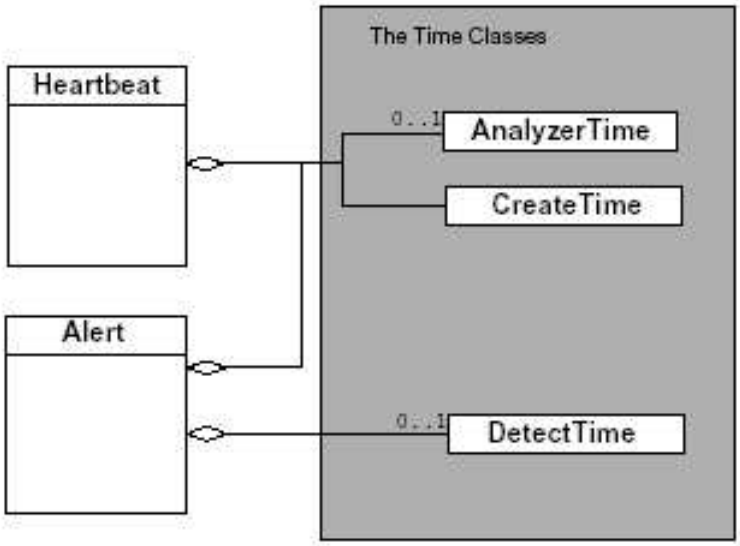

Figura 4.25: Le classi Time

Rango	Parola chiave	Descrizione
0	low	Bassa gravità
1	medium	Media gravità
2	high	Alta gravità

Tabella 4.4: Valori dell'attributo severity nella classe impact

4.2.3.18 Le classi Assessment

Il data model fornisce tre tipi di "accertamenti" che un sensore può fare a proposito di un evento. Queste classi sono aggregate della classe Assessment, come già visto in figura 4.24.

4.2.3.19 La classe Impact

Tale classe è usata per fornire al sensore una stima dell'impatto che l'evento ha sui suoi bersagli. La classe Impact ha tre attributi:

severity Una stima della gravità dell'evento. I valori permessi sono indicati in tabella 4.4:

Rango	Parola chiave	Descrizione
0	failed	Il tentativo non ha avuto successo
2	succeeded	Il tentativo ha avuto successo

Tabella 4.5: Valori dell'attributo completion nella classe impact

Rango	Parola chiave	Descrizione
0	admin	Si è cercato di ottenere (o si sono ottenuti) i privilegi come amministratore del sistema
1	dos	E' stato tentato oppure realizzato un DOS
2	file	E' stato tentata oppure realizzata un'azione su un file
3	recon	E' stata tentata oppure eseguita una ricongnizione
4	user	Si è cercato di ottenere o si sono ottenuti i privilegi come utente
5	other	Qualsiasi altra categoria non elencata precedentemente

Tabella 4.6: Valori attributo type nella classe impact

completion Indica se il sensore crede che il tentativo relativo all'evento registrato ha avuto successo oppure no. Ha solamente due valori permessi (vedi tabella 4.5):

type Il tipo di tentativo rappresentato dall'evento, raggruppato in ampie categorie. I valori permessi sono indicati in tabella 4.6. Il valore di default è `other`.

4.2.3.20 La classe Action

La classe **Action** è usata per descrivere ogni azione presa dal sensore per rispondere all'evento. L'unico attributo che presenta la classe è:

category Il tipo di azione presa. Il valore permessi sono elencati in tabella 4.7:

4.2.3.21 La classe Confidence

Questa classe viene utilizzata per rappresentare la miglior stima del sensore sulla validità delle sue analisi. La classe ha un solo attributo:

Rango	Parola chiave	Descrizione
0	block-installed	Un blocco di qualche tipo è stato installato per impedire ad un attacco di raggiungere il suo obbiettivo. Il blocco può riguardare porte, indirizzi, o addirittura disabilitare un account relativo ad uno specifico utente
1	notification-sent	Un messaggio di avviso viene spedito "fuori banda" (via pager, e-mail, etc.). Non include la trasmissione di questo allarme
2	taken-offline	Un sistema, un computer o un utente viene messo off-line
3	other	Qualsiasi altra categoria non elencata precedentemente

Tabella 4.7: Valori dell'attributo category nella classe action

Rango	Parola chiave	Descrizione
0	low	Il sensore dà una validità molto basse alle sue analisi
1	medium	Il sensore ritiene le sue analisi mediamente affidabili
2	high	Il sensore è sicuro del risultato delle proprie analisi
3	numeric	Il sensore fornisce la probabilità dell'affidabilità dei suoi risultati

Tabella 4.8: Valori dell'attributo rating nella classe confidence

rating La stima del sensore sulla sua validità nell'analizzare gli eventi. I valori permessi sono mostrati nella tabella 4.8; quello di default è `numeric`:

Questo elemento dovrebbe essere usato solo quando il sensore può produrre informazioni significative. I sistemi che hanno prodotto solamente un stima approssimativa sulla veridicità delle proprie analisi dovrebbero usare solamente `low`, `medium` o `high` come valore di stima.

4.2.3.22 Le classi Support

Tali classi costituiscono la maggior parte delle classi **Core** e sono tutte condivise tra loro.

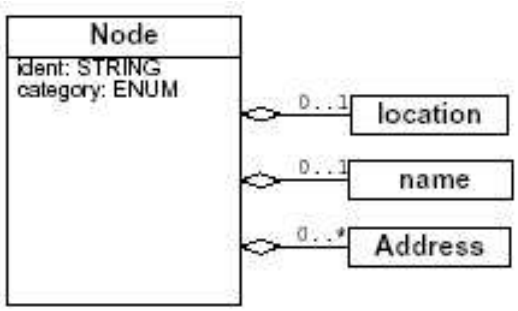

Figura 4.26: La classe Node

4.2.3.23 La classe Node

La classe **Node** è usata per identificare host e altri dispositivi di rete (router, switch, ecc.). Tale classe è composta da tre classi aggregate, come mostrato in figura 4.26.

Le classi aggregate che costituiscono la classe **Node** sono:

Location La locazione del host.

Name Il nome dell'host. Questa informazione deve essere fornita se l'informazione Address non viene data.

Address La rete o l'indirizzo hardware dell'host. Se il nome non viene fornito, essa è obbligatoria.

La classe **Node** ha due attributi:

ident Un identificatore univoco del nodo

category Il dominio dal quale è stata ricavata l'informazione relativa al nome. I valori permessi sono elencati in tabella 4.9. Il valore di default è `unknow`.

4.2.3.24 La classe Address

Questa classe è usata per rappresentare network, hardware e indirizzi di applicazioni. La classe **Address** è composta da due classi aggregate, come mostrato in figura 4.27:

Le classi aggregate che compongono la classe trattata sono:

Address L'informazione dell'indirizzo. Il formato di questo dato è governato dagli attributi di categoria.

Rango	Parola chiave	Descrizone
0	unknow	Dominio sconosciuto o irrilevabile
1	ads	Windows 2000 Advanced Directory Service
2	afs	Andrew File System (Transarc)
3	coda	Coda Distributed File System
4	dfs	Distributed File System (IBM)
5	dns	Domain Name System
6	hosts	Local host file
7	kerberos	Kerberos realm
8	nds	Novell Directory Services
9	nis	Network Information Services (Sun)
10	nisplus	Network Information Services Plus (Sun)
11	nt	Windows NT domain
12	wfw	Windows for Workgroups

Tabella 4.9: Valori dell'attributo category nella classe node

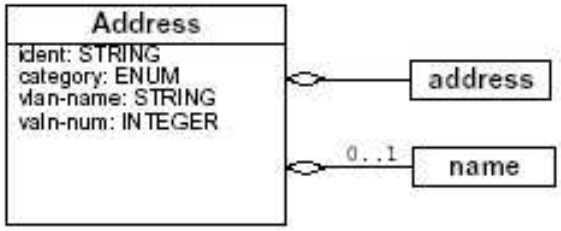

Figura 4.27: La classe Address

Rango	Parola chiave	Descrizione
0	unknow	Tipo di indirizzo sconosciuto
1	atm	Asynchronous Transfer Mode network address
2	e-mail	Electronic mail address (RFC 822)
3	lotus-notes	Lotus Notes e-mail address
4	mac	Media Access Control (MAC) address
5	sna	IBM Shared Network Architecture (SNA) address
6	vm	IMB VM ("PROFS") e-mail address
7	ipv4-net	IPv4 network address in dotted-decimal notation (a.b.c.d)
8	ipv4-addr-hex	IPv4 host address in hexadecimal notation
9	ipv4-net	IPv4 network address in dotted-decimal notation, slash, significant bits (a.b.c.d/nn)
10	ipv4-net-mask	IPv4 network address in dotted-decimal notation, slash, network mask in dotted-decimal notation (a.b.c.d/w.x.y.z)
11	ipv6-addr	IPv6 host address
12	ipv6-addr-hex	IPv6 host address in hexadecimal notation
13	ipv6-net	IPv6 network address, slash, significant bits
14	ipv6-net-mask	IPv6 network address, slash, network mask

Tabella 4.10: Valori dell'attributo category nella classe address

Netmask La network mask dell'indirizzo.

La classe **Address** ha quattro attributi:

ident Un identificatore univoco per l'indirizzo.

category Il tipo di indirizzo rappresentato. I valori permessi sono elencati in tabella 4.10. Il valore di default è `unknown`.

vlan-name Il nome della Virtual LAN al quale l'indirizzo appartiene.

vlan-num Il numero della Virtual LAN al quale l'indirizzo appartiene.

4.2.3.25 La classe User

La classe **User** è usata per descrivere gli utenti. E' principalmente usata come classe contenitore per la classe aggregata UserId, come si vede in figura 4.28.

La classe aggregata contenuta in **User** è:

UserId Identificatore di ciascun utente, come indicato dal suo tipo di attributo (vedi sezione 4.2.3.26).

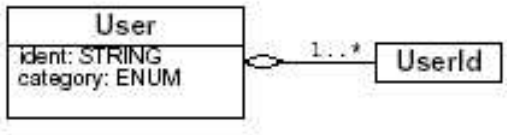

Figura 4.28: La classe User

Rango	Parola chiave	Descrizione
0	unknow	Tipo di utente sconosciuto
1	application	Utente dell'applicazione
2	os-device	Utente di un sistema operativo o di un device

Tabella 4.11: Valori dell'attributo category nella classe user

La classe **User** ha due attributi:

ident Identificatore univoco per l'utente

category Il tipo di utente rappresentato. I valori permessi sono elencati in tabella 4.11.

4.2.3.26 La classe UserId

La classe **UserId** fornisce informazioni specifiche su un utente. Più di un **UserId** può essere usato con la classe **User** per indicate tentativi di transizione da un utente ad un altro, o per fornire informazioni complete su i privilegi di un utente.

La classe **UserId** è composta da due classi aggregate, come mostrato in figura 4.29:

Le classi aggregate che costituiscono la **UserId** sono:

Name Il nome di un utente o di un gruppo.

Number Il numero di un utente o di un gruppo.

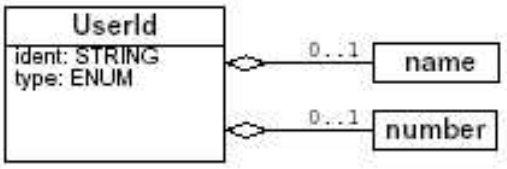

Figura 4.29: La classe UserId

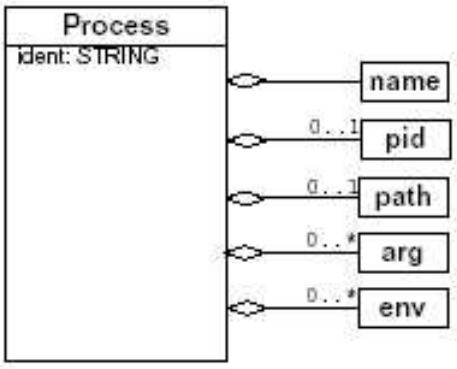

Figura 4.30: La classe Process

La classe **UserId** ha due attributi:

ident Un unico identificatore per l'utente.

type Il tipo di informazione sull'utente rappresentata. I valori permessi per questo attributo sono mostrati in tabella 4.12. Il valore di default è `original-user`.

4.2.3.27 La classe Process

Questa classe è usata per descrivere processi che sono stati eseguiti su sorgenti, bersagli e sensori. La classe **Process** è composta da cinque classi aggregate, come mostrato in figura 4.30.

Tali classi aggregate sono:

Name Il nome del processo che è stato eseguito. Questo è il nome corto; le informazioni sul path e sugli argomenti sono forniti altrove.

Pid L'identificatore del processo.

Path Il cammino completo del programma che è stato eseguito.

Arg La linea di parametri passati al programma. Possono essere specificati molteplici parametri (si assume nell'ordine in cui sono forniti) tramite l'uso ripetuto di arg.

Rango	Parola chiave	Descrizone
0	current-user	Lo user-id corrente che il processo o l'utente stanno utilizzando. In generale nei sistemi Unix, questo è lo user-id reale
1	original-user	L'attuale identificatore dell'utente o del processo che è stato riportato. Su sistemi che fanno qualche tipo di auditing o che operano estrazioni di un user-id dall"'audit-id token", questo valore può essere usato. Su quei sistemi che non supportano questo ma dove l'utente viene loggato nel sistema, potrebbe essere usato il "loggin id"
2	target-user	Lo user id dell'utente o del processo che si è tentato di diventare. Questo dovrebbe succedere quando un utente tenta di usare "su", "rlogin", "telnet", etc.
3	user-privs	Un altro user-id identifica l'utente o il processo che lo può utilizzare. Su sistemi Unix, questo sarebbe l'identificativo dell'utente per quanto riguarda i processi e i file. Gli user/id multipli possono essere usati per specificare un elenco di privilegi
4	current-groups	Il group id corrente (se applicabile) che è stato usato dall'utente o dal processo. Sui sistemi Unix, questo sarebbe il vero group id
5	groups-privs	Un altro group-id identifica l'utente o il processo che lo può utilizzare. Su sistemi Unix, questo sarebbe l'identificativo del gruppo per quanto riguarda i processi e i file. Sui sistemi BSD derivati da Unix, gli elementi user-id multipli di questo tipo sarebbero usati per includere tutti i group-id in un'apposita lista
6	other-privs	Non usato in un contesto utente, gruppo o processo, è usato solamente in un contesto di file. I permessi sui file assegnati ad utenti che non corrispondono né ai permessi degli utenti né ai permessi del gruppo. Sui sistemi Unix questo corrisponde ai permessi "world"

Tabella 4.12: Valori dell'attributo type nella classe userid

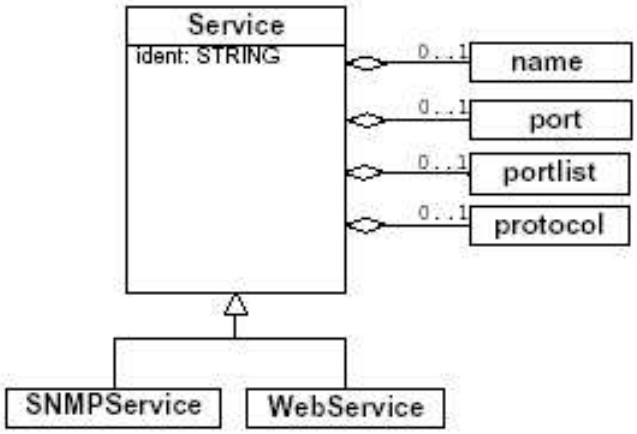

Figura 4.31: La classe Service

Env Una stringa di ambiente associato con il processo; generalmente nel formato `VARIABLE = value`. Anche qui la descrizione di molteplici stringhe di ambiente avvengono attraverso l'uso ripetuto di `env`.

4.2.3.28 La classe Service

Questa classe descrive i servizi di rete delle sorgenti e degli obiettivi. Può identificare i servizi dal nome, dalla porta e dal protocollo. Quando questa classe occorre come classe aggregata di **Source** (vedi sezione 4.2.3.10), è sottinteso che il servizio di una delle attività di interesse viene originata e che il servizio è collegato alle informazioni delle classi Node, Process e User tutte contenute in Source. Analogamente, quando **Service** occorre come classe aggregata della classe **Target** (vedi sezione 4.2.3.11), è sottinteso che il tipo di servizio in esame è analogo a quelli su cui viene diretta l'attività di interesse e che il servizio è collegato alle informazioni delle classi **Node**, **Process** e **User** tutte contenute in **Target**.

La classe **Service** è composta da quattro classi aggregate, come mostrato in figura 4.31.

Tali classi sono:

Name Il nome del servizio. Quando possibile, deve essere usato il nome dalla lista IANA[1] [IANA] delle porte conosciute.

[1]Internet Assigned Numbers Authority, è il coordinatore centrale per l'assegnazione dei valori di parametri univoci nei protocolli internet.

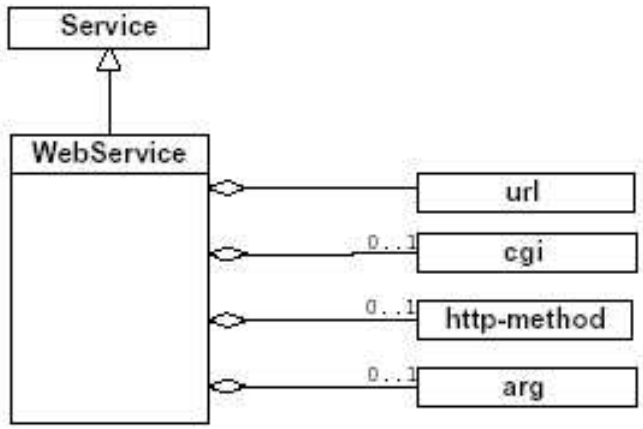

Figura 4.32: La classe WebService

Port Il numero della porta usata

Portlist Una lista dei numeri di porte usate.

Protocol Il protocollo usato.

Un elemento della classe **Service** deve essere specificato come nome o come porta o come lista di porte. Solamente queste combinazioni sono permesse.

4.2.3.29 La Classe WebService

Questa classe porta informazioni aggiuntive relative al traffico web. Le quattro classi che compongono la classe **WebService** sono mostrate in figura 4.32.

Url L'URL presente nella richiesta.

Cgi Lo script CGI presente nella richiesta, senza argomenti.

Http-method Il metodo HTTP (PUT, GET) usato nella richiesta.

Arg I parametri passati allo script CGI.

4.2.3.30 La classe SNMPService

Questa classe porta informazioni aggiuntive relative al traffico SNMP ed è composta da tre classi aggregate, come mostrato in figura 4.33.

Le classi aggregate che costituiscono la classe **SNMPService** sono:

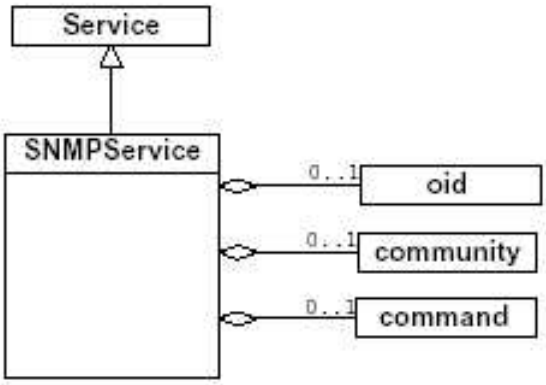

Figura 4.33: La classe SNMPService

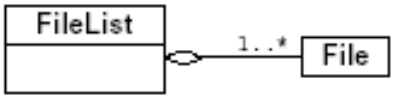

Figura 4.34: La classe FileList

Oid L'identificatore del'oggetto.

Community La stringa community dell'oggetto, se il traffico usa i protocolli SNMPv1 o SNMPv2c.

Command Il comando spedito al SNMP server (GET, SET, ecc.).

4.2.3.31 La classe FileList

Questa classe descrive file e altri oggetti trattati come file appartenenti ai target. E' fondamentalmente usata come classe contenitore della classe aggregata **File**, come mostrato in figura 4.34.

La classe aggregata contenuta in **FileList** è:

File Informazione sul singolo file come indicato dalla sua category e dagli attributi fstype (vedi sezione 4.2.3.32).

4.2.3.32 La classe File

La classe **File** fornisce specifiche informazioni su file o altri oggetti trattati come file

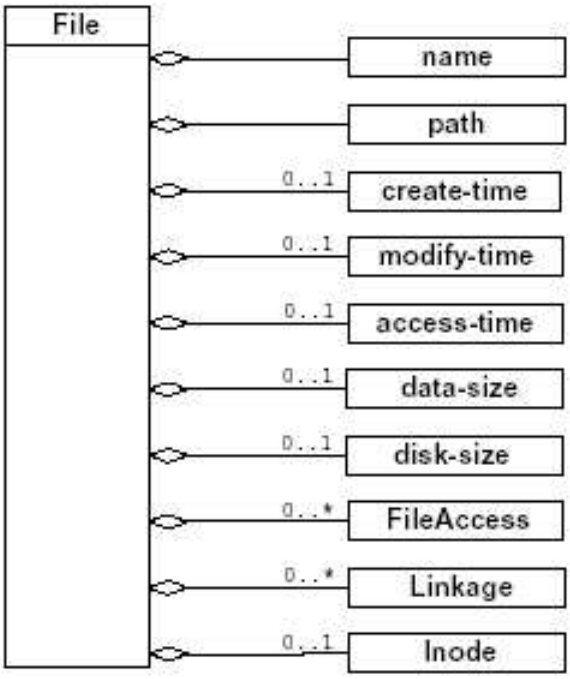

Figura 4.35: La classe File

che sono stati creati, cancellati, o modificati sul target. La descrizione può fornire sia il file setting precedente l'evento rivelato sia il file setting nel momento stesso dell'evento rilevato, come specificato usando l'attributo *category*.

La classe **File** è composta da dieci classi aggregate, come mostrato in figura 4.35.

Nome Il nome del file al quale l'allarme è collegato, senza il suo path.

Path Il path completo del file, incluso il nome. Il path dovrebbe essere mostrato nel modo più standard possibile, per facilitare il processo di generazione dell'allarme.

Create-time L'ora in cui il file è stato creato.

Modify-time L'ora in cui il file è stato modificato per l'ultima volta.

Access-time L'ora in cui si è accesso al file per l'ultima volta.

Disk-size Lo spazio fisico occupato dal file espresso in bytes.

FileAccess I permessi di accesso al file.

Rango	Parola chiave	Descrizione
0	current	L'informazione sul file è stata raccolta dopo aver apportato i cambiamenti
1	original	L'informazione sul file è relativa alla versione originale

Tabella 4.13: Valore dell'attributo category nella classe file

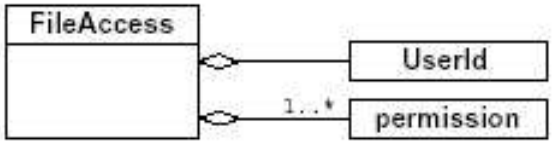

Figura 4.36: La classe FileAccess

Linkage Gli oggetti del file system ai quali questo file è linkato.

Inode Informazioni dell'inode per questo file.

La classe **File** ha solamente tre attributi:

xident E' l'identificatore del file.

category Il contesto nel quale è stata fornita l'informazione. I valori permessi sono elencati in tabella 4.13. Non esiste nessun valore di default.

fstype Il tipo di file system in cui il file risiede. Il nome deve essere specificato usando le abbreviazioni standard. Questi attributi stabiliscono come il path e gli altri attributi vengono interpretati.

4.2.3.33 La classe FileAccess

Tale classe rappresenta i permessi di accesso su un file. Le due classi aggregate che la compongono sono mostrate in figura 4.36.

UserId L'utente o il gruppo al quale questi permessi vengono applicati. I valori per il tipo devono essere `user-privs`, `group-privs` o `other-privs`. Altri tipi di valori non devono essere usati in questo contesto.

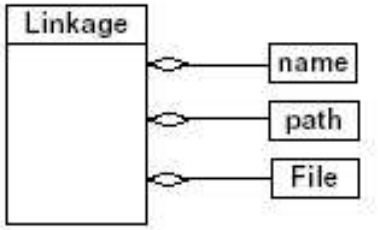

Figura 4.37: La classe Linkage

Permission Il livello id accesso consentito. I valori raccomandati sono `noAccess`, `read`, `write`, `execute`, `delete`, `executeAs`, `changePermission` e `takeOwnerShip`. Le stringhe `changePermission` e `takeOwnerShip` rappresentano questi concetti in Windows. In Unix, il proprietario del file ha sempre l'accesso a `changePermission`, anche se nessun altro accesso è permesso per quell'utente.

4.2.3.34 La classe Linkage

La classe **Linkage** rappresenta le connessioni del file system tra il file descritto nell'elemento File e gli altri oggetti nel file system. Per esempio, se l'elemento File è un link simbolico o una scorciatoia, allora l'elemento Linkage dovrebbe contenere il nome dell'oggetto linkato.

La classe suddetta è composta da tre file aggregati, come mostrato in figura 4.37. Tali classi sono:

Name Il nome dell'oggetto del file system, path escluso.

Path L'intero cammino dell'oggetto considerato, incluso il nome. Tale path dovrebbe essere rappresentato nella forma più standard possibile onde facilitare l'analisi dell'allarme.

File L'elemento File può essere usato al posto degli elementi name e path se l'informazione aggiuntiva sul file è inclusa.

La classe **Linkage** ha un solo attributo:

category Il tipo di oggetto che il link descrive. I valori permessi sono elencati in tabella 4.14:

Rango	Parola chiave	Descrizione
0	hard-link	L'elemento <name> rappresenta un altro nome per questo file. Questa informazione può essere più facilmente ottenibile su file system NTFS che altrove
1	mount-point	Un alias per la directory specificata dal <name> e dal <path> del padre
2	reparse-point	Applicata solo su Windows; esclude i link simbolici e i punti di mount, che sono tipi specifici di reparse point
3	shortcut	
		Il file rappresenta per Windows una "scorciatoia". Una scorciatoia si distingue da un link simbolico nella differenza del loro contenuto, il quale può essere particolarmente importante essere importante per il manager
4	stream	Un Alternate Data Stream (ADS) in Windows; una fork su MacOs. L'entità separata del file system che è considerata un'estensione del File principale
5	symbolic-link	L'elemento name rappresenta il file al quale l'elemento punta.

Tabella 4.14: Valori dell'attributo category nella classe linkage

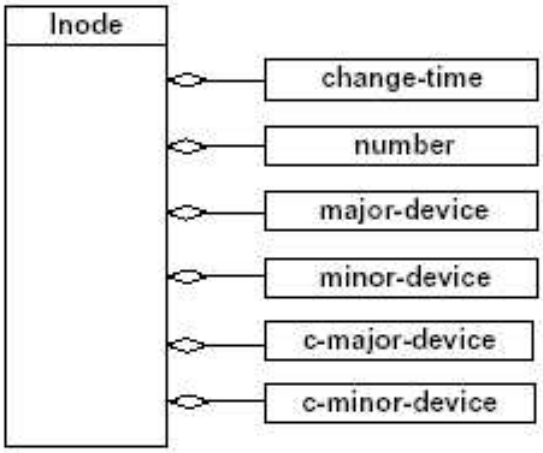

Figura 4.38: La classe Inode

4.2.3.35 La classe Inode

Questa classe è usata per rappresentare le informazioni aggiuntive contenute in un i-node del file system di Unix. La classe **Inode** è composta da sei classi aggregate, come mostrato in figura 4.38.

Change-time L'ora del cambiamento dell'ultimo inode.

Number Il numero dell'inode.

Major-device Il numero major-device del dispositivo in cui il file risiede.

Minor-device Il numero minor-device del dispositivo in cui il file risiede.

C-major-device Il major-device dello stesso file se esso è un device speciale per i caratteri.

C-minor-device Il minor-device dello stesso file se esso è un device speciale per i caratteri.

Notare che **Number**, **Major-device**, e **Minor-device** devono essere forniti assieme, così come **C-major-device** e **C-minor-device**.

Capitolo 5

Modello esperto di IDS

Nei capitoli 3 e 4 si sono descritte in dettaglio le potenzialità di un sistema esperto e di un sistema ibrido per la rilevazione delle intrusioni all'interno di una rete. Lo scopo di questa tesi è quello di fondere tali potenzialità creando un sistema IIDS (Intelligent IDS) capace di decidere in modo autonomo il tipo di reazini da adottare a fronte di qualsiasi attacco rilevato. Le motivazioni che hanno spinto a sviluppare tale integrazione verranno descritte in sezione 5.1, mentre nella sezione 5.2 verrà descritta la fase di analisi teorica del sistema, specificando quali tipologie di allarme sono state oggetto di studio.

Prima di entrare nel dettaglio del lavoro svolto è doveroso chiarire le ipotesi che si sono fatte all'inizio della fase di progettazione. Anzitutto bisogna specificare che l'obbiettivo di questa tesi non è quello di produrre un sistema di difesa di reti locali commercializzabile, ma quello di creare una versione modificata del sistema Prelude IDS in modo da integrarlo con il sistema esperto CLIPS. In fase si progetto si è quindi ipotizzato di poter disporre di tutti i dati necessari all'IIDS per raggiungere una determinata decisione; ciò significa che si è supposto che all'interno della LAN ove è installato, il sistema possa interfacciarsi con sensori in grado di fornire tali dati. In secondo luogo si è presupposto di poter disporre di una serie di applicazioni che implementano tutte le contromisure previste dal sistema; infine si è deciso di tralasciare l'implementazione delle chiamate a tali applicazioni, creando un'architettura che lasci la possibilità futura di sviluppare tale integrazione.

5.1 Sistemi esperti ed IDS

Come anticipato precedentemente, in questa sezione vengono illustrate le motivazioni che sono alla base dello sviluppo dell'Intelligent IDS S.L.U.N.P. (Sinthetic Lifeform Userd for Network Pacekeeping). Consideriamo anzitutto lo scenario rappresentato dalla figura 5.1.

Il tipico scenario rappresentato da una LAN in cui viene installato Prelude IDS: una serie di sensori vengono installati nei vari punti di interfacciamento tra la rete locale ed il mondo esterno; tali sensori comunicano con un server centrale che immagazzina i dati e li rende disponibili off-line grazie ad un database che tiene traccia di tutti gli eventi giudicati sospetti dai sensori. L'utente (generalmente l'amministratore della sicurezza) può verificare solo peridicamente lo stato della rete, stato che di per sé non può giudicarsi attuale poiché rappresenta solamente un'istantanea di una realtà in continuo mutamento. E' facile capire come questa situazione non sia assolutamente sostenibile se la rete locale che deve essere protetta ricopre un'importanza elevata (si pensi ad esempio ai server della Banca d'Italia oppure alla rete dell'amministrazione pubblica) poiché gli hacker hanno forti interessi a superare i suoi sistemi di sicurezza al fine di persegiure i loro scopi illeciti.

Un primo metodo per ovviare a questo grave handicap è rappresentato dalla creazione di un'intefaccia real-time che comunichi immediatamente la presenza di un nuovo all'arme all'utente. Tale interfaccia può inoltre suggerire all'utente quali possibili contromisure si possono adottare in base ai diversi tipi di attacco rilevati. Questo scenario è rappresentato in figura 5.2.

Indubbiamente, questa risoluzione elimina il problema di comunicare in modo tempestivo all'amministratore della sicurezza il verificarsi di un attacco; tuttavia bisogna effettuare un'analisi approfondita dei vantaggi e degli svantaggi che questa soluzione presenta:

- Vantaggi
  - Permette all'utente di avere un quadro chiaro e preciso di ciò che accade nella rete locale in qualsiasi istante
  - Non si introducono tempi di attesa per la validazione da parte dell'utente delle decisioni prese rigurado le contromisure da attivare poiché è egli stesso a falro

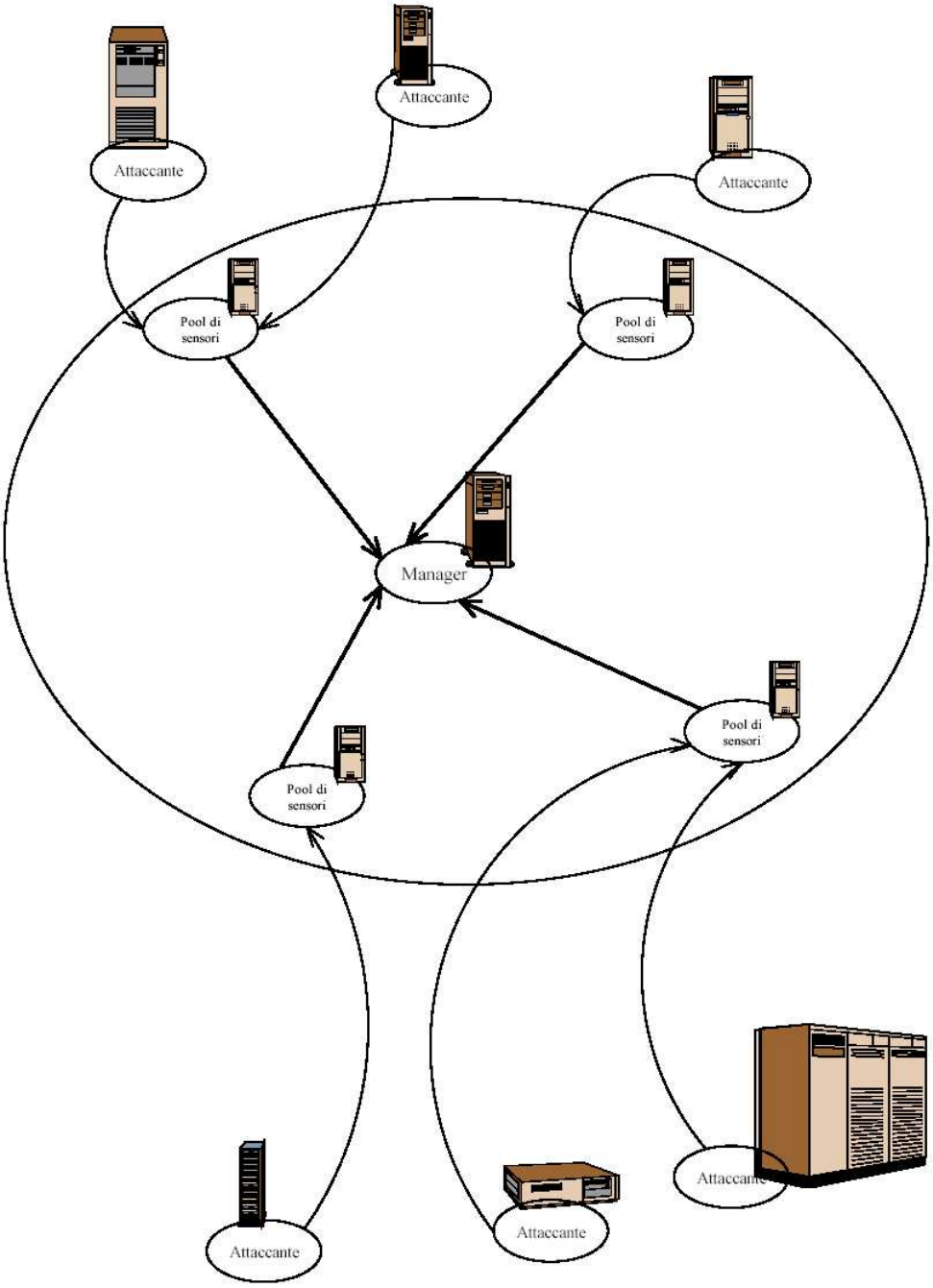

Figura 5.1: Tipico scenario di utilizzo di Prelude IDS

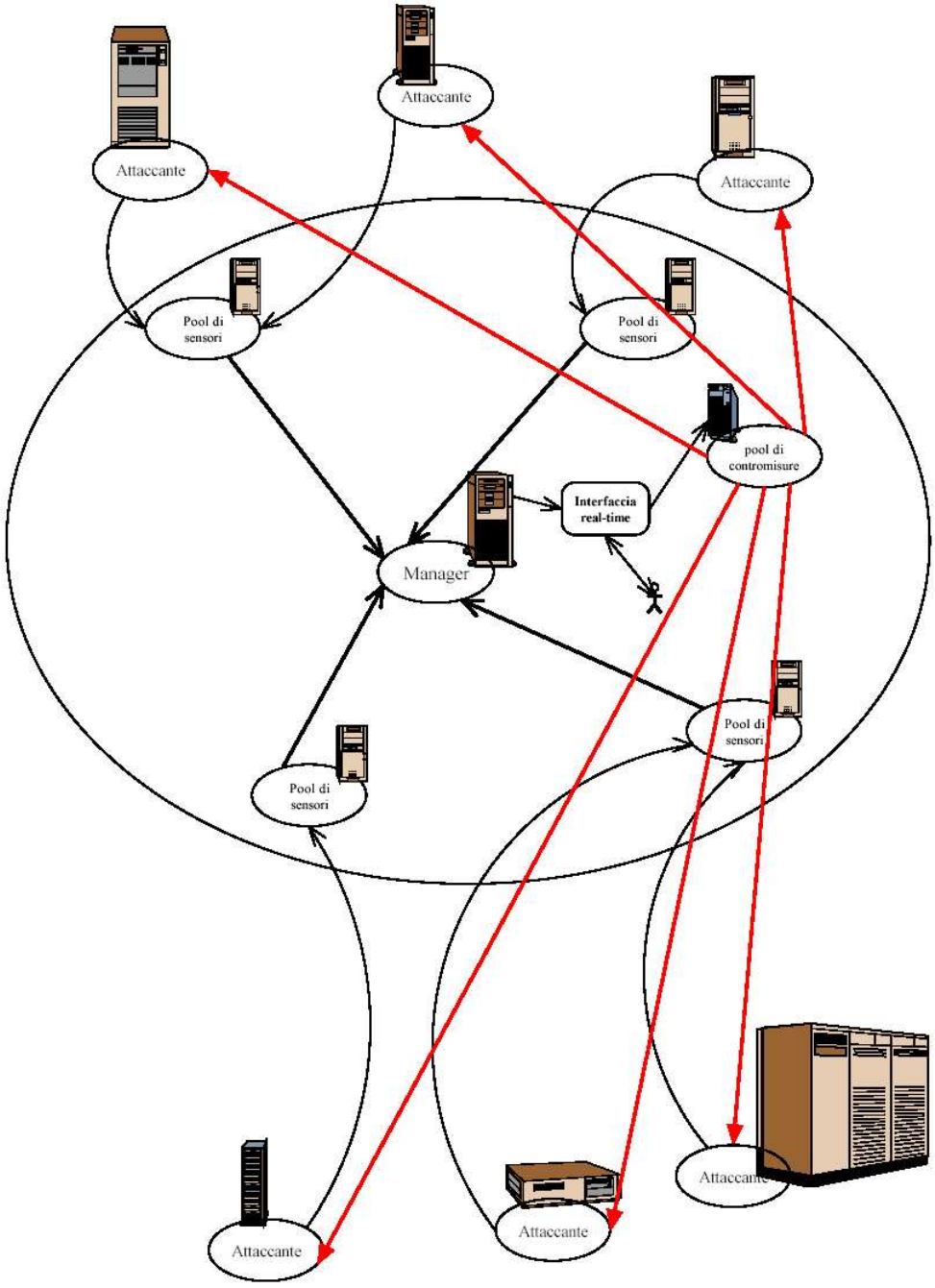

Figura 5.2: Prelude IDS ed interfaccia real-time

  - L'utente può visionare immediatamente tutti i dettagli relativi all'attacco rilevato
  - In caso di una tipologia di attacco molto pericolosa, l'amministratore della sicurezza è in grado di avvertire tutti gli utenti della lan entro breve tempo dalla rilevazione dell'attacco

- Svantaggi
  - Poiché per ogni allarme rilevato viene visualizzata una serie di possibili contromisure da adottare, l'utente è obbligato a conoscere a fondo i meccanismi che sono alla base dell'allarme stesso e di tutte le contromisure adottabili
  - Per il precedente motivo questo tipo di sistema può essere adottato solamente da chi ha a disposizione un personale altamente qualificato e costantemente aggiornato sugli sviluppi degli attacchi e delle contromisure
  - Se si vuole reagire ad un attacco è necessario che vi sia sempre almeno un utente che monitori l'interfaccia di Prelude
  - In caso di attacco distribuito (più sorgenti che attaccano contemporaneamente lo stesso obbiettivo) o nel caso in cui si registrino molti attacchi (di tipo differente) contemporanei, l'utente è impossibilitato ad analizzare e reagire in breve tempo ad ogni attacco, con il risultato che aumenta in modo notevole la possibilità di riuscita delle azioni illecite rilevate
  - Vi è la possibilità che l'utente commetta degli errori di valutazione dovuti a distrazione o a scarsa conoscenza di un particolare attacco o di una particolare contromisura

Uno scenario alternativo a quello introdotto precedentemente prevede di modificare Prelude IDS rendendolo capace di sfruttare un sistema esperto al fine di scegliere automaticamente le contromisure maggiormente adatte a contrastare i diversi tipi di attacco rilevati. Questo scenario è rappresentato in figura 5.3.

Prima di attivare contromisure particolarmente onerose a livello di risorse di sistema o che prevedono l'attivazione di operazioni irreversibili, l'IIDS può richiedere l'assenso dell'amministratore della sicurezza poiché deve essere sua la responsabilità dell'attuazione delle reazioni maggiormente drastiche. Per questo motivo è prevista

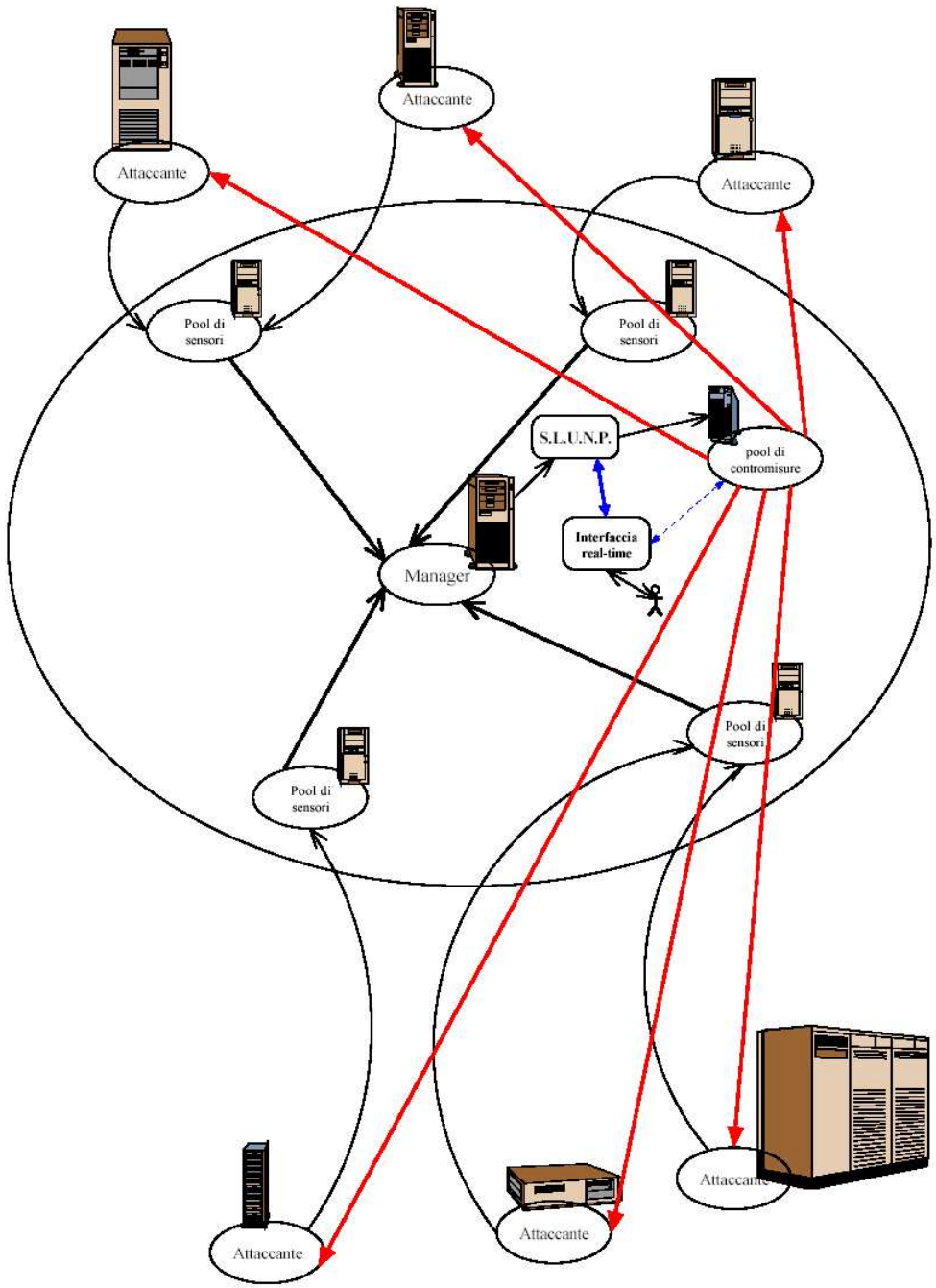

Figura 5.3: Prelude IDS e sistema decisionale SLUNP

un'intefaccia utente real-time che fornisce un rapporto su ciò che è stato rilevato nella rete e su ciò che il sistema esperto ha deciso di fare, quindi viene richiesto all'utente un semplice avvallo delle decisoni prese. Se l'utente è d'accordo l'IIDS procede in linea con quanto ha deciso, altrimenti lascia il pieno controllo all'utente che attiverà manualmente le contromisure rituenute più opportune.

Andiamo ora ad analizzare i vantaggi e gli svantaggi di questo secondo scenario:

- Vantaggi
  - Non è più necessaria la presenza costante di un utente che aspetti il verificarsi di un attacco per decidere le relative reazioni
  - L'utente non è più obbligato a conoscere ogni dettaglio di tutti gli attacchi e di tutte le possibili contromisure
  - In caso si verifichi un elevato numero di attacchi in un lasso di tempo molto breve (anche in meno di un minuto), il sistema riesce comunque a gestire tutte le situazioni con un ritardo minimo fra rilevazione dell'azione illegale ed attivazione delle relative contromisure
  - Qualora vengano corretti tutti i bug che il sistema decisionale presenta, la probabilità di prendere una decisione errata si riduce ad una frazione minima
  - l'IIDS può essere installato all'interno di qualsiasi realtà, dal piccolo negozio che ha pochi computer in rete alla grande impresa con migliaia di pc connessi fra loro
- Svantaggi
  - Ogni volta che si riceve notizia dello sviluppo di un nuovo tipo di attacco o di una nuova contromisura il sistema decisionale deve essere aggiornato
  - Si introducono ritardi quando le decisioni prese dall'Intelligent IDS devono essere validate dall'utente prima di poter dare inizio alla reazione (questo si può verificare quando le contromisure risultano molto onerose per il sistema o quando prevedono azioni drastiche come, ad esempio, la rimozione dalla lan di un particolare utente)
  - Se un intruso riesce a compromettere l'integrità del sistema decisionale l'intera rete locale potrebbe risultare totalmete vulnerabile a qualsiasi attacco

Dopo aver effettuato un'attenta valutazione di entrambi i possibili scenari, il migliore è risultato essere il secondo, cioè quello che prevede lo sviluppo di un IDS che, in collaborazione con un sistema esperto, è in grado di ditinguere quali siano le contromisure più adeguate per fronteggiare gli attacchi a chi è sottoposta la rete locale. Le motivazioni che hanno spinto a realizzare questo sistema sono molteplici, anche se molte di esse sono insite nei vantaggi che il sistema è in grado di apportare: innanzitutto il fattore determinante che è stato preso in considerazione è rappresentato dal *tempo di reazione* all'allarme, tempo che viene calcolato dall'istante in cui viene rilevato l'attacco stesso all'istante in cui viene attivata la prima contromisura decisa dal sistema SLUNP. Considerando gli esperimenti condotti (vedi capitolo 7) possiamo affermare che di norma il tempo di reazione rimane inferiore ai dieci secondi e su di esso incide in maniera preponderante la velocità di trasferimento del messaggio IDMEF di allarme (vedi sezione 4.2) dal sensore al manager (il sistema esperto impiega pochi millisecondi per raggiungere la propria decisione). Tale tempo di reazione si alza di diversi ordini di grandezza (può essere anche di diverse ore) qualora SLUNP debba aspettare la conferma dell'amministratore della sicurezza prima di attivare le contromisure identificate. E' facile capire come la classificazione di gravità delle contromisure adottate possa incidere sul tempo di reazione del sistema agli attacchi: se la conferma da parte dell'utente è prevista solamente per una piccola percentule di contromisure relative ad attacchi di particolare gravità, il tempo di reazione sarà sempre minimo poiché sarà il sistema esperto a prendere la maggior parte delle decisioni. Viceversa, se la percentuale di reazioni che necessitano di approvazione è molto elevata si rischia di annullare quasi del tutto l'azione del sistema esperto, ricadendo di fatto nello scenario proposto in figura 5.2, con tutti i suoi vantaggi e svantaggi.

In particolare, se consideriamo la situazione rappresentata in tale figura, un utente incontra diverse difficoltà nel gestire più allarmi rilevati contemporaneamente (sopratutto se il numero di tali allarmi risulta essere molto alto) ed inevitabilmente il tempo di reazione si innalza rendendo al rete vulnerabile, annullando di fatto l'utilità del sistema di contromisure installato.

Un'altra motivazione che ha fatto sì di scartare il primo scenario è l'osservazione che il livello di concentrazione di un essere umano non può essere costante (dipende dallo stato fisico della persona e dai fattori ambientali del luogo di lavoro) e questo porta inevitabilmente a commettere degli errori. Prendere decisioni errate in termini

di sicurezza della rete potrebbe portare a conseguenze irreparabili; è per questo che si è voluto introdurre un sistema automomo che, dopo un periodo di transizione necessario per eliminare gli eventuali bug di programmazione, manterrà un grado di efficienza costante e giungerà alla medesima decisone ogni volta che i dati in ingresso avranno un determinato valore, indipendentemente da qualsiasi fattore ambientale.

Sono queste le motivazioni che hanno mosso la realizzazione del sistema SLUMP di cui parleremo nelle sezioni successive.

5.2 Modellazione del sistema esperto realizzato

In questa sezione viene trattata in dettaglio la progettazione teorica dell'IIDS. Per aumentare la portabilità e la riutilizzabilità del software si è deciso di realizzare un sistema composto da vari moduli, ognuno dei quali si occupa di una ben determinata tipologia di allarmi. In particolare, sono stati previsti cinque diversi livelli decisionali autonimi fra di loro. La scelta di suddividere il problema decisionale in più livelli permette di ridurre la complessità dei percorsi decisionali (e quindi i tempi di decizione) ed inoltre riduce la quantità righe di codice da aggiornare ogni volta che si viene a conoscenza di un nuovo tipo di attacco non ancora conosciuto dal sistema. A tale scopo si sono analizzate nel tettaglio le signature delle regole utilizzate dal sistema di intrusion detection SNORT^{TM}[SNORT], regole che vengono utilizzante anche dai sensori Prelude NIDS e Prelude LML. Quest'analisi ha permesso di creare 23 tipologie di attacco che permettono di isolare il livello base del sisema esperto dai livelli superiori: qualora si venga a conoscenza di una nuova signature relativa ad un nuovo attacco, basterà modificare il livello base del sistema esperto inserendo la nuova signature in una tipologia di attacco già esistente. In questo modo l'aggiornamento interesserà solamente uno dei livelli decisionali riducendo il tempo ed i costi di upgrade.

Le tipologie di attacco sono mostrate nelle tabelle 5.1 e 5.2.

Prima di passare a descrivere in dettaglio i livelli ed i moduli che costituiscono la struttura di del progetto SLUNP è necessaria una precisazione sugli schemi che verranno introdotti in seguito. Questi schemi non sono tradizionali diragrammi di flusso dei dati che vengono utilizzati per rappresentare lo schema del progetto del software, ma piuttosto di *alberi decisionali* che hanno il compito di mostrare quali domande il sistema esperto si ponga per giungere ad una decisione una volta che

Tipologia	Descrizione
Worm	Raggruppa tutti gli attacchi di tipo worm
Mail-bomb	Raggruppa tutti gli attacchi che sfruttano la posta elettronica
Nuke	Raggruppa tutti gli allarmi di tipo nuke, come ad esempio l'attacco *Ping of Death*
Backdoor	Raggruppa tutti gli attacchi che creano una backdoor per superare le difese del sistema attaccato
Port-scan	Raggruppa tutti gli attacchi che effettuano un port scan sulla rete (richieste su porte anomale, apertura di una porta normalmente non utilizzata dai servizi della lan, probing di rete, etc.)
Sovraccarico	Raggruppa tutti gli attacchi e tutti gli eventi che provocano un sovraccarico all'interno del sistena: dal sovraccarico di CPU o hard disk su un singolo host al sovraccarico della LAN stessa.
Ddos	Raggrupa tutti gli attacchi DoS di tipo distribuito
Buffer overflow	Raggruppa tutti gli attacchi che sfruttano l'errata gestione del buffer overflow
Accesso file protetti	Raggruppa tuttti gli attacchi che sono condotti al fine di modificare file protetti (modifica della tabella di routing, defacement del sito internet, etc.)
Utente loggato come altro utente	Raggruppa tutti gli attacchi che permettono di effettuare un login all'interno della rete locale con una falsa identità (shell remota di roothijacking, etc.)
Indirizzo spoofed	Raggruppa tutti gli attacchi che sono generati qualora si identifichi un pacchetto con indirizzo IP sorgente spoofed e che non rientrano nelle altre tipologie di attacco
Bonk	Raggruppa tutti gli attacchi di tipo *Bonk*
Attesa	Particolare tipo di allarme che viene utilizzato da SLUNP per effettuare attivazioni fra vari livelli. Questa tipologia di attacco non raggruppa altri tipi di allarmi
Storico	Particolare tipo di allarme che viene utilizzato da SLUNP per generare allarmi da inserire nel database di Prelude IDS. Questa tipologia di attacco non raggruppa altri tipi di allarmi

Tabella 5.1: Tipologie di attacchi previste in SLUNP

Tipologia	Descrizione
Attacco sconosciuto	Questa tipologia di attacco viene utilizzata da SLUNP per indicare che ciò che sta avvenendo nella rete locale è dipeso da un evento che non è riconducibile a nessun tipo di attacco conosciuto dal sistema stesso
Frammentazione pacchetti ip	Raggruppa tutti gli attacchi che tentano di violare le difese della rete locale sfruttando gli errori prodotti dall'erraga gestione dei pacchetti IP frammentati (fragment overrun, etc.)
Violazione ICMP	Raggruppa tutti gli attacchi che hanno come scopo la violazione di un database e che non rientrano nella tipologia *SQL injection*
Spamming	Raggruppa tutti gli allarmi generati dallo spamming effettuato tramite posta elettronica
SQL injection	Raggruppa tutti gli allarmi che tentano di effettuare query illegali in databade SQL (sfruttando form presenti su pagine html, etc.)
Teardrop	Raggruppa tutti gli attacchi di tipo Teardrop
Inizio attacco	Questa tipologia di attacco viene utilizzata da SLUNP per indicare l'elevata probabilità che la rete stia per essere attaccata
Rpc	Raggruppa tutti gli attacchi di tipo RPC
Violazione DB	Raggruppa tutti gli attacchi che hanno come scopo la violazione di un database e che non rientrano nella tipologia *SQL injection*

Tabella 5.2: Tipologie di attacchi previste in SLUNP

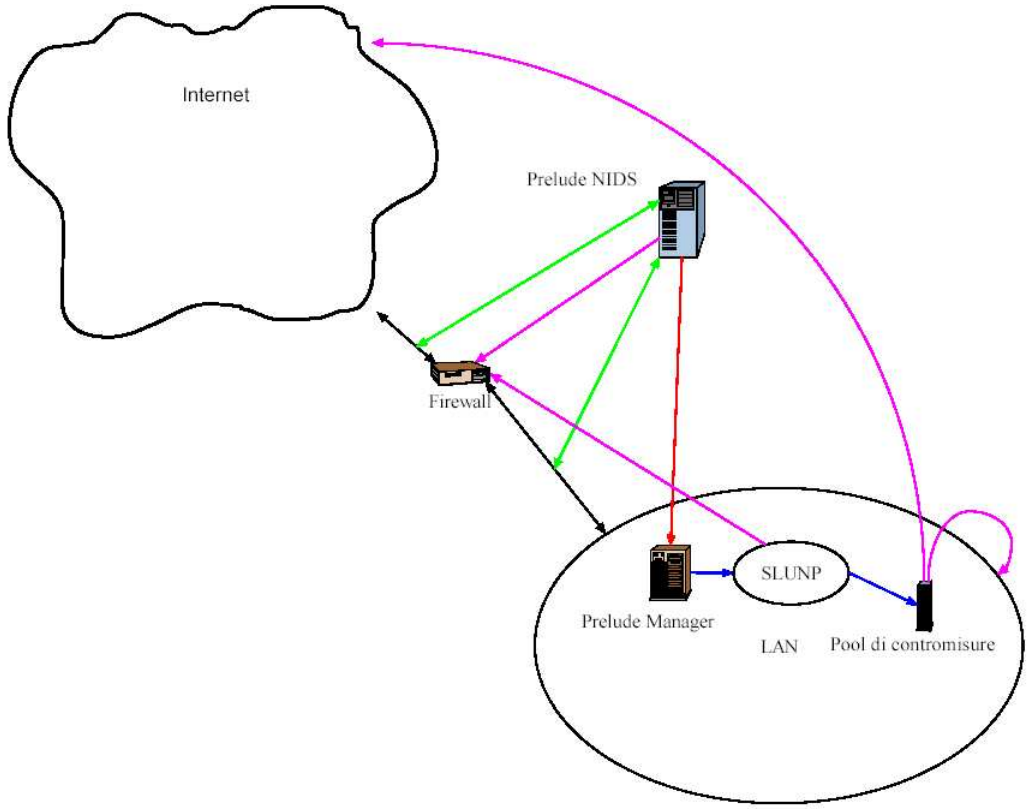

Figura 5.4: Schema dello schenario utilizzato nella seconda fase della progettazione di SLUNP

riceva i dati necessari in ingresso. Per facilitare la comprensione di tali alberi ogni schema verrà accompagnato da un'ampia descrizione che ne illustrerà tutti gli aspetti fondamentali.

E' inoltre necessaria una modellazione maggiormente dettagliata dello scenario che si è assunto studiare: se infatti è sufficiente la figura 5.3 per giustificare le scelte fatte a livello di prima analisi progettuale, lo schema in figura 5.4 ci consentirà di comprendere meglio alcuni passaggi ed alcune condizioni che sono presenti negli alberi decisionali descritti di seguito.

Qui si vede come il sensore di rete Prelude NIDS catturi i pacchetti del traffico diretto verso il firewall e proveniente da esso. Qualora il sensore rilevi un evento anomalo viene generato un allarme verso Prelude Manager ed eventualmente si agisce direttamente sul firewall. Il manager attiva il sistema esperto SLUNP che può eventualmente annullare le azioni del sensore sul firewall prima di attivare le

contromisure più adatte a fronteggiare il tipo di attacco rilevato.

Chiarito questo ultimo punto possiamo passare alla descrizione dettagliata degli alberi decisionali di SLUNP.

5.2.1 Livello 0: primo livello decisionale di SLUNP

I livelli decisionali in cui è stata suddivisa la struttura di SLUNP sono cinque ed il Livello 0 rappresenta il primo livello che viene chiamato in causa ogni volta che il manager di Prelude IDS riceve comunicazione di un evento anomalo da parte dei sensori. L'albero decisionale relativo a questo livello è rappresentato in figura 5.5.

Quando il manager riceve un allarme da uno dei sensori installati nella rete locale attiva il Livello 0 il quale sfrutta le conoscenze che ha sono presenti nella base delle conoscenze (vedi sezione 2.1) per determinare se per tale allarme il sensore abbia la possibilità di operare direttamente sul firewall attivando una prima contromisura. Se il sensore non può operare in questo modo, il sistema esperto utilizza le regole presenti all'interno del Modulo 01 (vedi sezione 5.2.2) per giungere ad una decisione; qualora invece il sensore possa interagire direttamente con il firewall il sistema SLUNP utilizza l'interfaccia real-time per effettuare un rapporto completo all'utente e per chiedergli di avvallare la contromisura attuata dal sensore. Se l'utente decide che la contromisura del sensore non è corretta, SLUNP provvederà ad attivare il tool più appropriato per annullare la reazione attuata dal sensore, dopo di ché seguirà l'albero decisionale presente all'interno del Modulo 01. Se l'utente decide invece di mantenere attiva la contromisura presa, SLUNP richiede all'utente se vuole attivare altre reazioni per l'allarme rilevato o se la reazione presa sia sufficiente. Se così è SLUNP termina la sua esecuzione, altrimenti individuerà le altre contrimisure grazie al Modulo 01.

5.2.2 Modulo 01

L'albero decisionale di questo modulo è rappresentato in figura 5.6.

Per prima cosa SLUNP verifica che il *confidence rating* stabilito dal sensore (vedi sezione 4.2.3.21) sia accettabile, cioè sia maggiore o uguare al livello minimo giudicato accettabile dall'amministratore della sicurezza (durante la fase implementativa descritta nella sezione 6 tale livello è stato settato a 0.5). Se il livello non è accettabile SLUNP termina l'esecuzione del Livello 0 ed attiva il Livello 0b che rappresenta

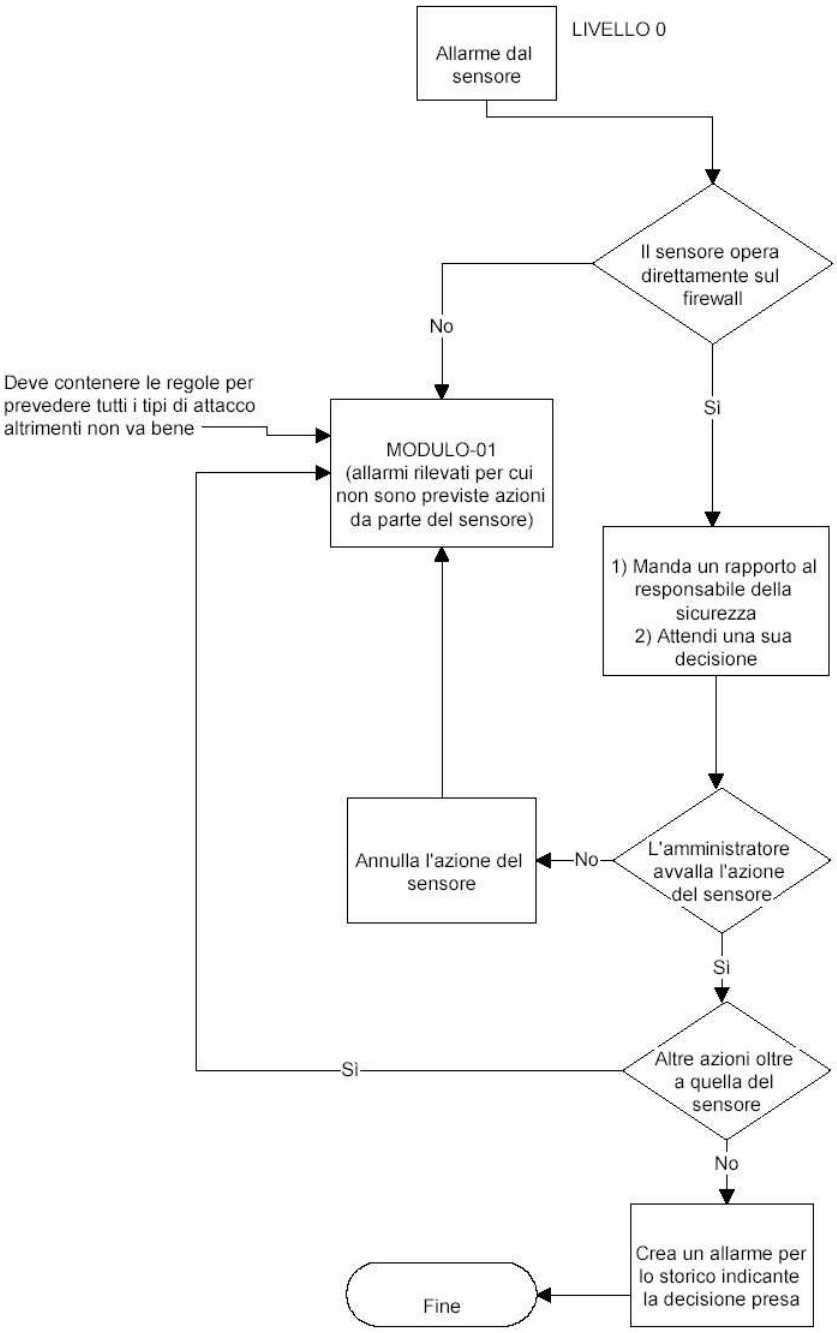

Figura 5.5: Albero decisionale del Livello 0

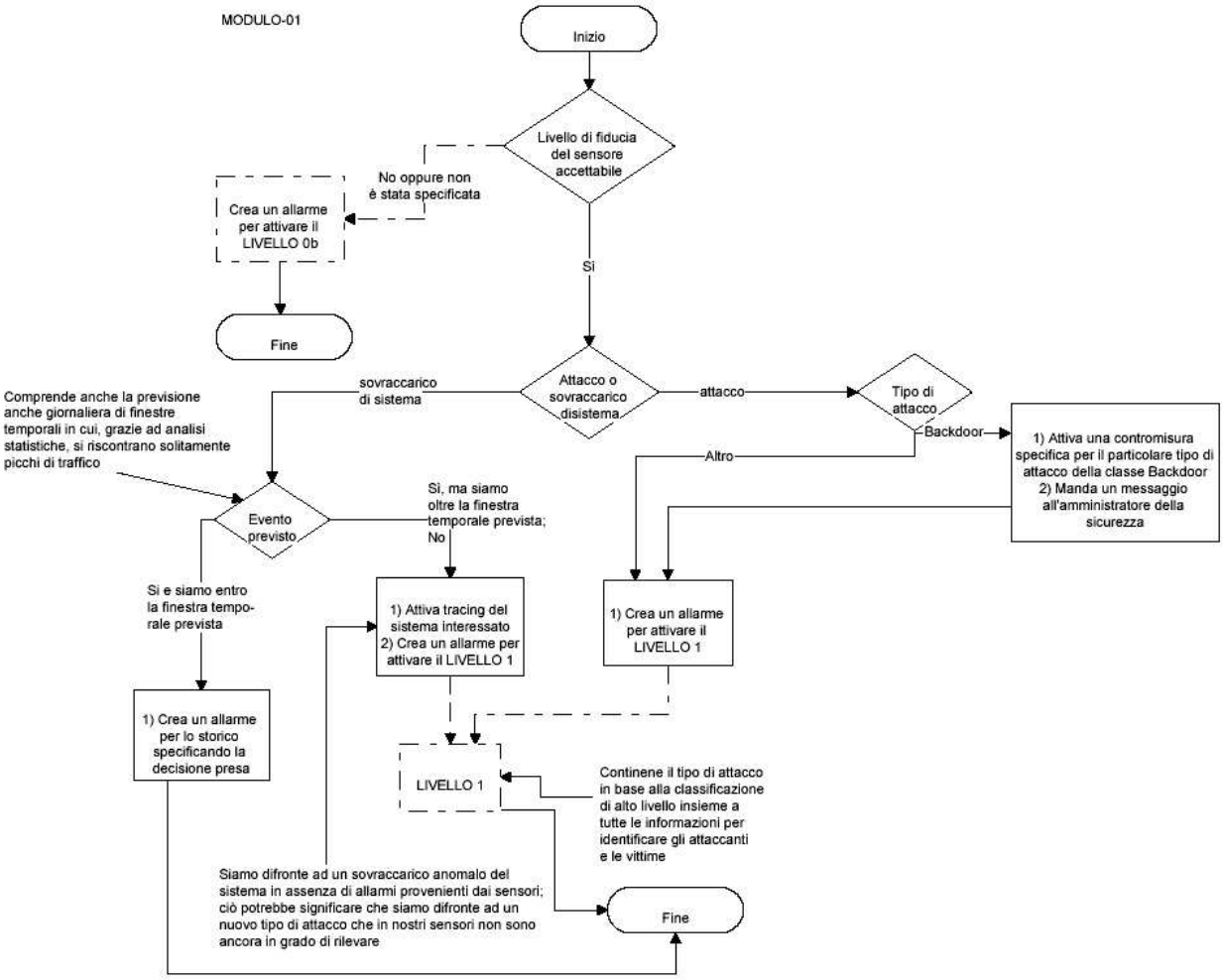

Figura 5.6: Albero decisionale del Modulo 1

il secondo dei cinque livelli che compongono il sistema esperto realizzato (vedi sezione 5.2.3). Se il livello di confidenza dell'allarme è giudicato sufficientemente alto si valuta se l'evento anomalo rappresenta un attacco conosciuto o un sovraccarico di sistema.

Se si tratta di un attacco conosciuto si valuta se appartiene alla tipologia Backdoor, nel qual caso si attiva una contromisura specifica per il particolare tipo di attacco e si manda un messaggio sull'interfaccia real-time; quindi si crea sempre un allarme che ha il compito di attivare il Livello 1 (vedi sezione 5.2.4), per qualsiasi tipo di attacco conosciuto.

Se si tratta di un sovraccarico di sistema, SLUNP verifica se tale evento era previsto: consideriamo ad esempio un'agenzia giornalistica che riesca ad ottenere i diritti di trasmissione delle olimpiadi (o solamente di determinati eventi sportivi olimpici) tramite streaming video attraverso internet. E' facile comprendere come nelle fasce orarie in cui si svolgono le gare il server che ha il compito di trasmettere lo streaming video sia sommerso di richieste, generando una serie di allarmi di sovraccarico che in realtà sono dei *falsi positivi*, poiché il sovraccarico non è determinato da un tentativo di attacco di tipo Dos o DDos, ma dall'attività dell'agenzia giornalistica. Un altro esempio può essere rappresentato da una società che, tramite datamining, ha stabilito come il proprio sito sia oggetto di un picco di richieste in una determinata fascia oraria. Considerando questi esempi è facile immaginare le conseguenze che un'analisi errata dei dati sugli accessi alla rete da parte degli utenti esterni potrebbe comportare: il rischio è quello di considerare attività sgradite richieste di servizi fatte in fasce orarie che non sono state considerate oppure considerare operazioni normali tentativi di DoS condotti in periodi temporali giudicati erroneamente di interesse da parte degli utenti esterni alla LAN.

Ritornando all'albero decisionale, se il sovraccarico è stato previsto, SLUNP crea un allarme che Prelude IDS memorizzerà nel proprio database, altrimenti verrà attivato un tool che effettuerà una registrazione di tutte le attività che si svolgono nel sistema vittima del sovraccarico. Ciò ha due utilità: consente di avere prove legali da esibire in futuri dibattimenti e fornisce dati che potrebbero rivelarsi utili al fine di identificare un nuovo tipo di attacco non ancora conosciuto da SLUNP. Infine viene creato un allarme di attivazione per il Livello1, dopo di ché l'esecuzione di questo livello decisionale termina.

5.2.3 Livello 0b: secondo livello decisionale di SLUNP

Questo livello decisionale ha il solo scopo di incrementare il valore del tasso di confidenza dell'allarme ricevuto. L'albero decisionale del Livello 0b è rappresentato in figura 5.7.

Ricevuto l'allarme dal Livello 0, SLUNP procede all'attivazione di tools specifici per ogni tipologia di attacco che hanno il compito di effettuare rilevamenti sulla rete locale ed operazioni di datamaining sul databse al fine di poter fornire al sistema esperto i dati necessari per decidere se l'allarme originale generato dal sensore apparteneva alla tipologia corretta (in questo caso viene restituito al Livello 0 l'allarme originale con un valore di confidence rating pari a 0.9) oppure no, nel qual caso viene generato un nuovo allarme relativo alla tipologia corretta avente confidence rating pari a 0.9. E' interessante notare il fatto che è la chiamata ai tools di analisi è di tipo iterattivo, nel senso che se un determinato tool non è riuscito a raggiungere il livello di confidenza minimo, SLUNP ha la possibilità di attivarne altri per aumentare la precisione delle analisi e giungere ad una decisione definitiva.

5.2.4 Livello 1: terzo livello decisionale di SLUNP

Questo è il terzo livello decisionale di SLUNP; il suo albero decisionale è mostrato in figura 5.8.

L'allarme proveniente dal Livello 0 viene analizzato per determinare a quale tipo di attacco si riferisce. Ottenuta l'informazione cercata, il sistema esperto attiva direttamente una serie di contromisure (nel caso di attacco *Bonk*, *IP overlapping*, *Buffer overflow*, *Teardrop*) oppure segue l'albero decisionale presente in uno dei moduli che compongono questo livello decisionale. E' importante notare come in questo livello decisionale venga presa in considerazione l'importanza del server che è sotto attacco (vedi sezione 5.1): in base ad essa verranno considerati diversi alberi decisionali che porteranno all'attivazione, qualora avvenga, di diverse contromisure per una stessa tipologia di attacco.

5.2.5 Modello sovraccarico 1

Questo albero decisionale (mostrato in figura 5.9 e con maggior dettaglio in figura 5.10 e 5.11) viene utilizzato per stabilire se l'allarme generato dal sensore è relativo ad un sovraccarico di sistema.

Per prima cosa si verifica se la rete esterna (quella che commette i gateway ed i

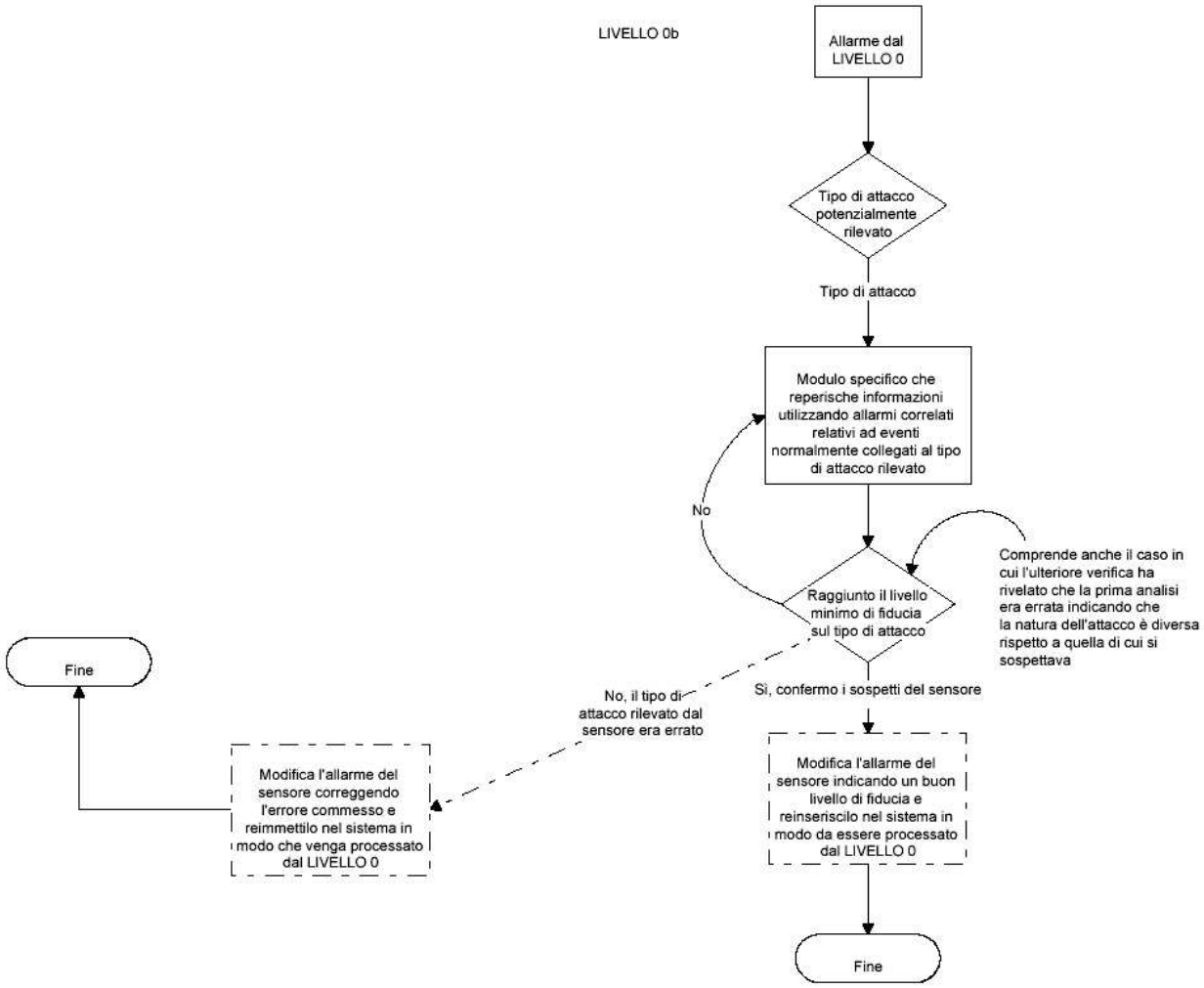

Figura 5.7: Albero decisionale del Livello 0b

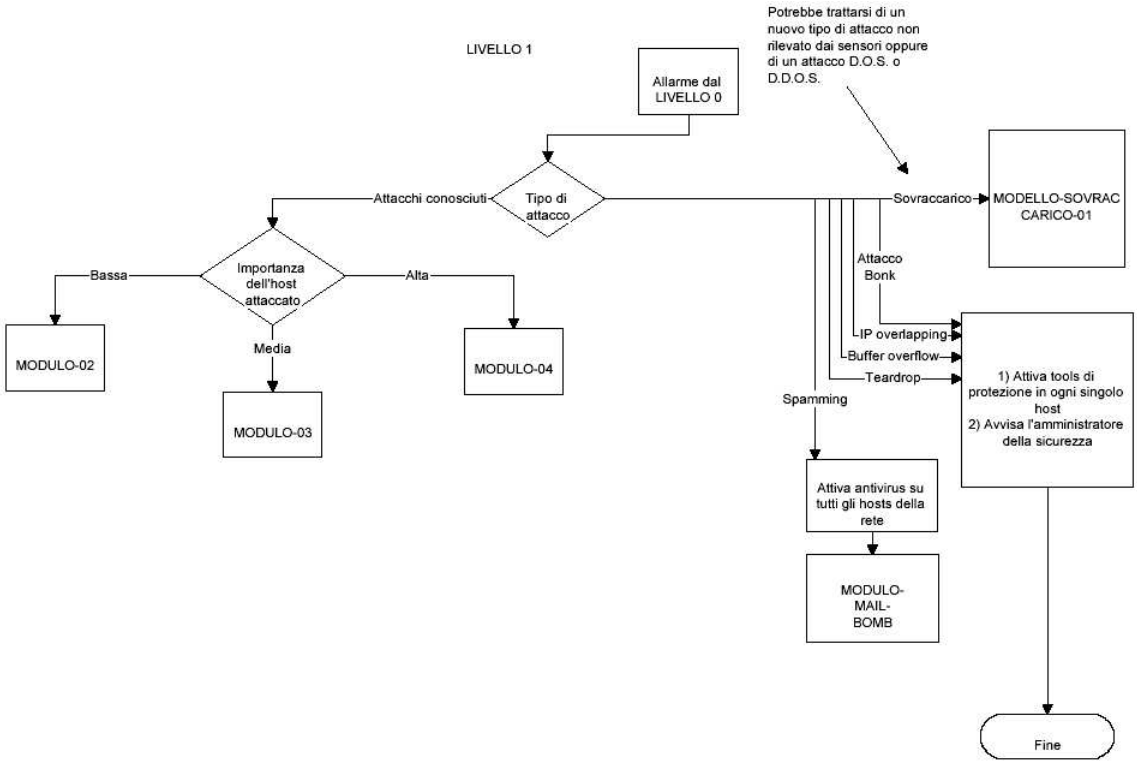

Figura 5.8: Albero decisionale del Livello 1

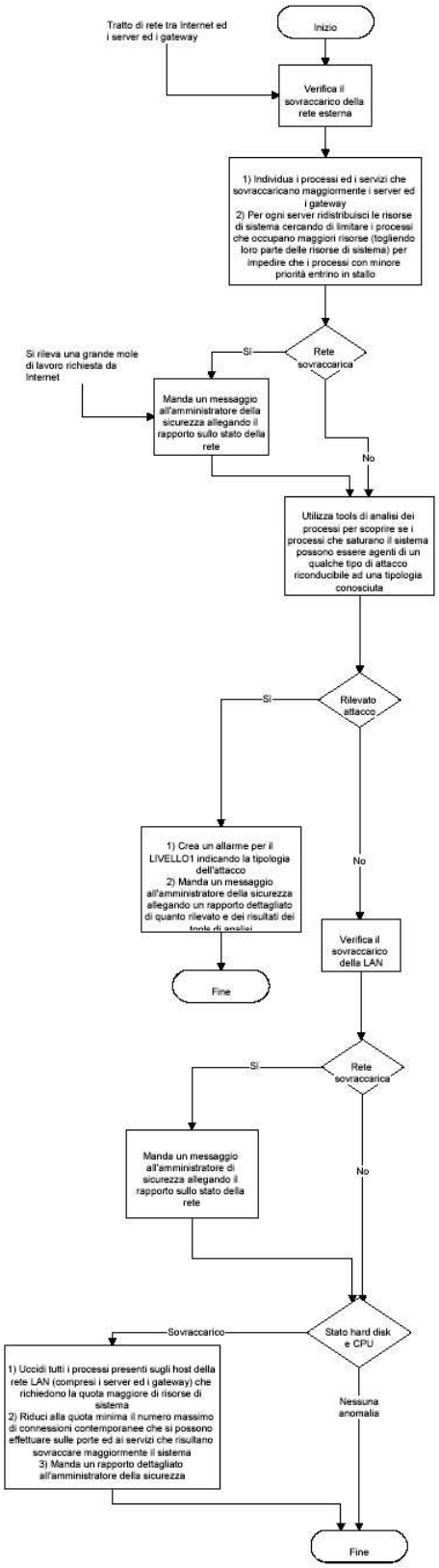

Figura 5.9: Albero decisionale del modello sovraccarico 1

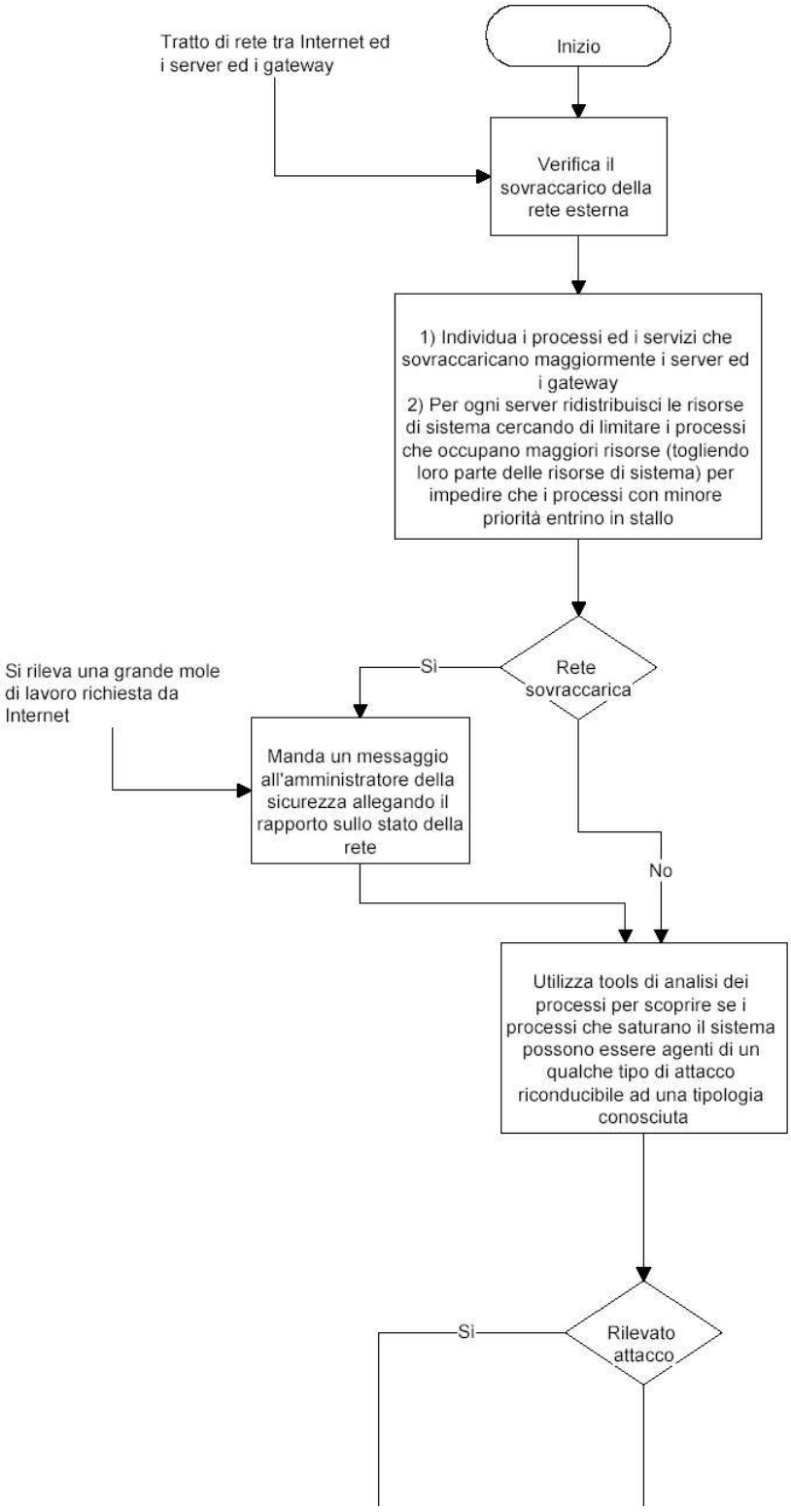

Figura 5.10: Albero decisionale del modello socraccarico 1 (prima parte)

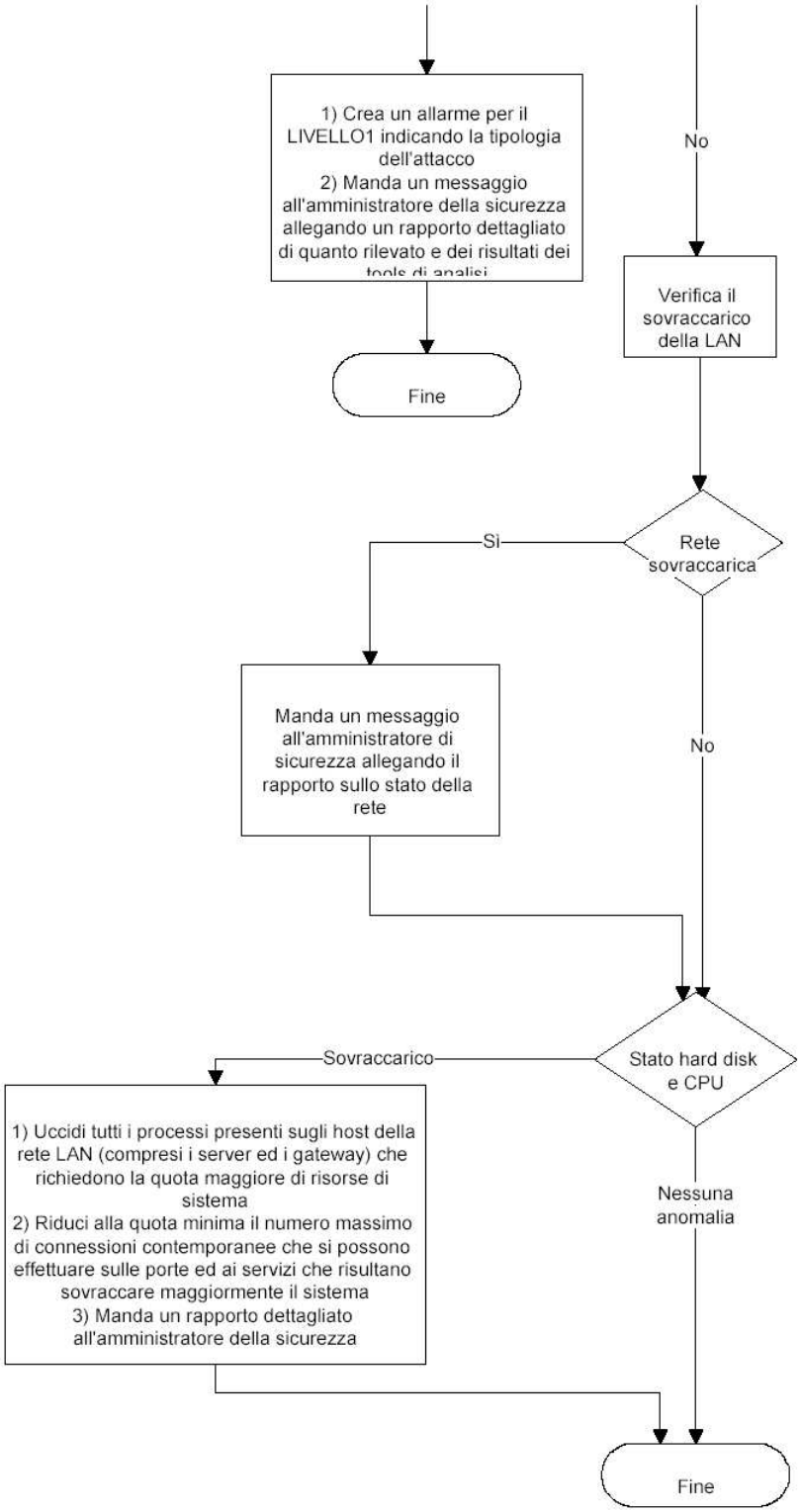

Figura 5.11: Albero decisionale del modello socraccarico 1 (seconda parte)

server con Internet) è sovraccarica. Vengono quindi attivati tools che controllano lo stato dei singoli server e gateway, indipendentemente dallo stato della rete esterna. Queste contromisure permettono di individuare i processi ed i servizi che sovraccaricano maggiormente quei sistemi in modo da sottrarre loro parte delle risorse di sistema allocate per tentare di ristabilire una situazione accettabile.

Viene quindi preso in considerazione il risultato della verifica effettuta sulla rete esterna. Se tale rete risulta essere sovraccarica l'unica cosa che il sistema esperto è in grado di fare è di avvertire l'amministratore della sicurezza; in ogni caso vengono attivati tool di analisi dei processi attivi su tutti i sistemi saturi al fine di individuare eventuali processi che vengono utilizzati normalmente in uno degli attacchi conosciuti. Se questa verifica ha esito positivo viene creato un nuovo allarme per il Livello 1 che specifica la vera natura dell'attacco ed inoltre viene mandato un rapporto dettagliato all'utente, dopo di ché l'esecuzione di questo modulo termina.

Se invece la verifica ha esito negativo si procede ad attivare un tool di analisi dello stato della lan. Se risulta sovraccarica viene mandato un rapporto dettagliato all'amministratore della sicurezza.

Vengono quindi presi in considerazione lo stato degli hard disk e della CPU dell'host interessato dal sovraccarico. Se si riscontra un sovraccarico significa che tutte le misure prese sinora non hanno avuto gli esiti sperati e quindi si procede all'uccisione dei processi e dei servizi (questi vengono successivamente riavvitati per impedire il verificarsi di un DoS) che determinano il sovraccarico; tale procedura viene effettuata su tutti gli host della rete, compresi i server ed i getewai. Viene quindi ridotto il numero massimo di connessioni contemporanee che possono utilizzare i servizi precedentemente uccisi ed infine viene mandato un rapporto dettagliato all'amministratore della sicurezza.

5.2.6 Modulo mailbomb

L'albero decisionale in figura 5.12 rappresenta i processi logici che il sistema esperto segue qualora venga rilevato un attacco di tipo mailbomb.

Come specificato nello schema, questo tipo di attacco prevede la spedizione di un grandissimo numero di mail; tale attacco può essere rilevato solamente all'interno della rete locale e le contromisure adottabili si riducono sostanzialmente ad una sola: impedire all'utente locale responsabile dell'attacco di spedire qualsiasi e-mail. Sebbene quest'azione sembri ledere i diritti dell'utente, non si deve dimenticare il

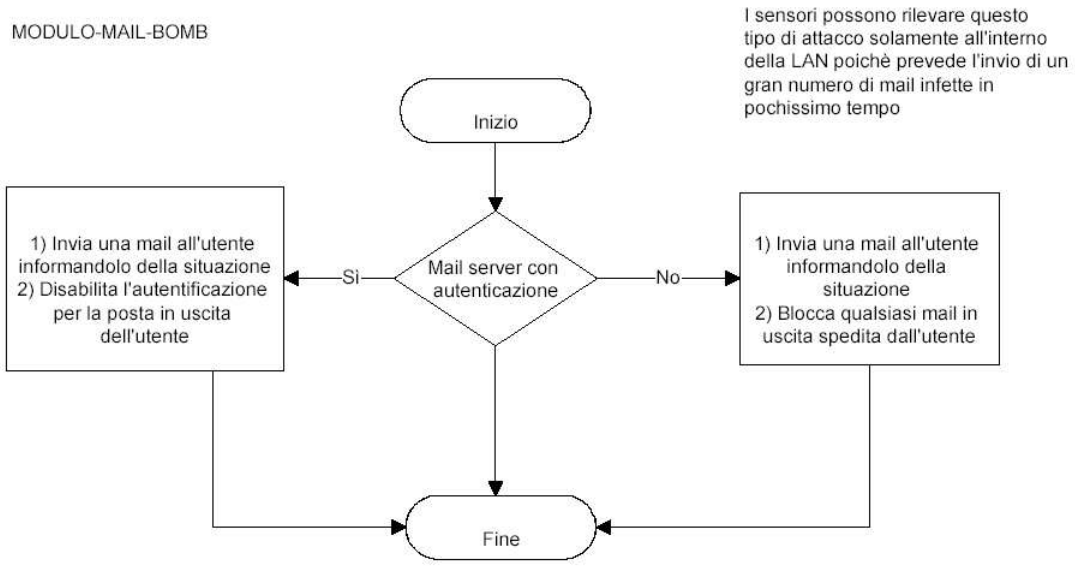

Figura 5.12: Albero decisionale del modulo mailbomb

fatto che l'utilizzatore di un account della rete deve osservare delle regole di comportamento correto, sia sia nei confronti degli utenti interni che esterni. Se ciò non accade, se cioè l'utente utilizza il suo account a fini illeciti l'amministratore della sicurezza è autorizzato a prendere tutti i provvedimenti ritenuti utili a far cessare tale attività.

5.2.7 Modulo 02

Questo modulo decisionale viene utilizzato qualora l'importanza del server attaccato sia bassa; ad esempio una piccola impresa che ha deciso di gestire localmente la propria pagina internet di e-commerce. Il termine "bassa" non è certo riferito alla realtà locale dell'azienda (per la quale il server ha un'importanza altissima), ma alla più vasta e complessa realtà di Internet (vedi sezione 5.1). L'albero decisionale è mostrato in figura 5.13.

In base alla tipologia di attacco segnalata dal Livello 0 verranno attivate delle contromisure specifiche (è questo il caso degli allarmi del tipo *RPC*, *ICMP*, *SQL injection*, *accesso file protetti*), verranno seguiti alberi decisionali presenti in altri moduli, verranno creati nuovi allarmi (attacco di tipo *DDoS*) oppure non verrà prevista alcuna contromisura (attacco *Port scan* e *Backdoor*).

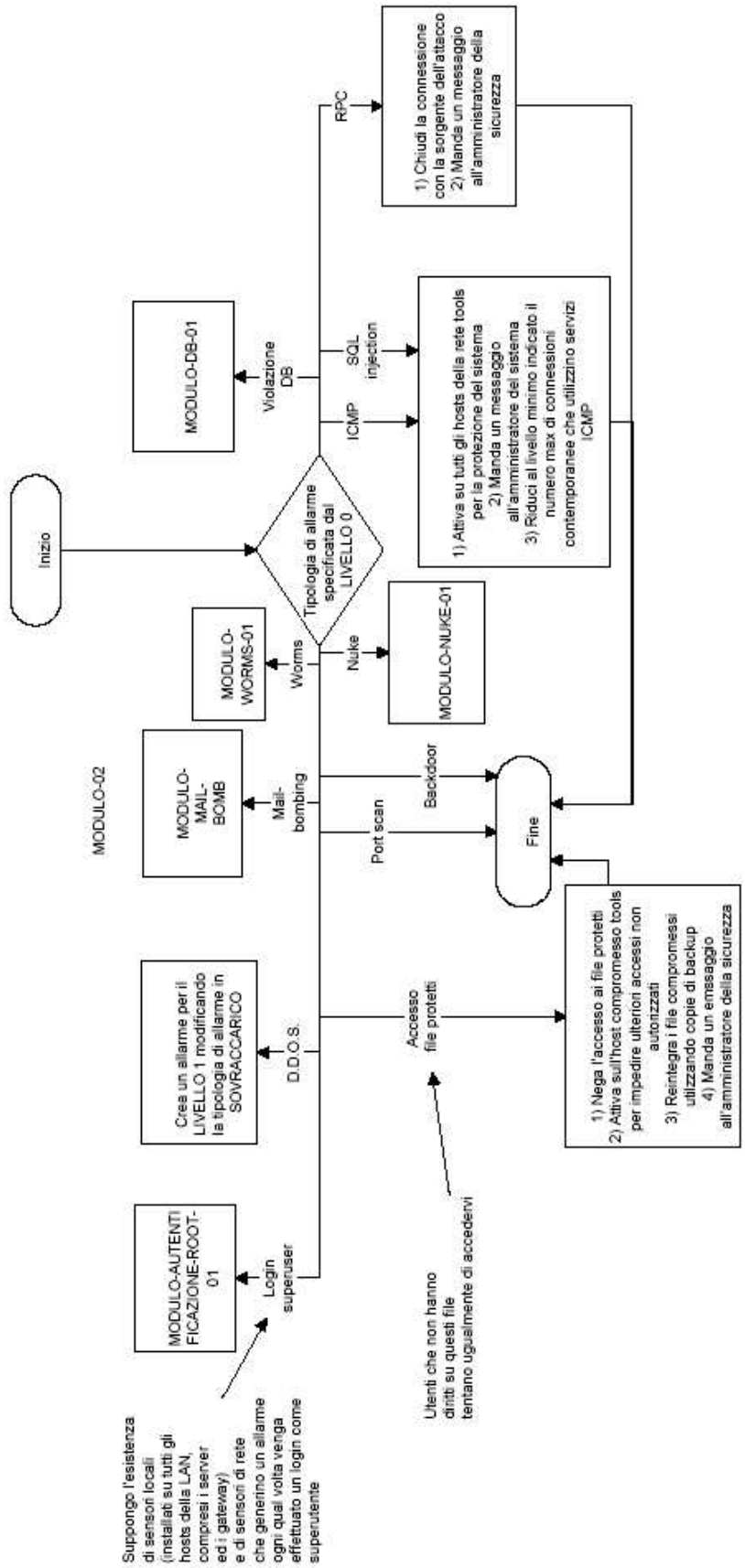

Figura 5.13: Albero decisionale del modulo 02

5.2.8 Modulo DB 1

L'albero decisionale di figura 5.14 mostra come la prima contromisura adottata da SLUNP sia quella di chiudere la connessione al database che ha provocato l'allarme.

Dopo aver effettuato questa operazione, che ha lo scopo principale di proteggere il database da ulteriori pericoli di manomissione o di furto di dati, viene analizzato l'allarme per determinare se la sorgente dell'attacco era interna alla rete (traffico uscente) oppure esterna (traffico entrante).

Se il traffico risulta essere uscente significa che c'è un utente della rete locale che sta compiendo azioni illecite e quindi il sistema è autorizzato a prendere tutti i provvedimenti ritenuti necessari. Innanzitutti si verifica se la sorgente dell'attacco è localizzata su una workstation oppure su un server. nel primo caso l'azione compiuta dall'utente è talmente grave da giustificare un riavvio forzato dell'host, anche se ciò può creare danni potenziali anche ad altri utenti della rete. Se la sorgente dell'attacco è situata su un server questo livello decisionale non è ingrado di procedere oltre e quindi crea un allarme per il successivo livello decisionale fornendogli tutti i dati in suo possesso. In ogni caso, prima di terminare l'esecuzione, questo modulo manda un rapporto dettagliato all'amministratore della sicurezza.

Se il traffico risulta essere entrante significa che la sorgente dell'attacco si trova al di fuori della rete locale. Poiché l'inportanza del server è giudicata bassa, il sistema può attivare solamente contromisure di analisi del flusso di pacchetti di rete allo scopo di rintracciare la macchina attaccate ed effettuare un rapporto dettagliato all'amministratore della sicurezza.

5.2.9 Modulo nuke 01

Come mostrato in figura 5.3 la prima cosa che SLUNP controlla è l'esistenza o meno di allarmi rivolti al Livello 2 (vedi sezione 5.2.25) relativi alla stessa tipologia di attacco, stessa sorgente e stesso obbiettivo.

Questo controllo è necessario poiché, data la gravità dell'attacco e l'importanza del server, si è optato per una linea di difesa poco aggressiva: prima di attivare le contromisure a fronte dell'attacco si vuole attendere un certo periodo di tempo (in genere X minuti); se in tale lasso di tempo non si registrano nuovi attacchi non si procede oltre, altrimenti si crea un allarme per lo storico indicando che esistono già allarmi rivolti al livello 02. Giunti a questo punto è doverosa un'anticipazione

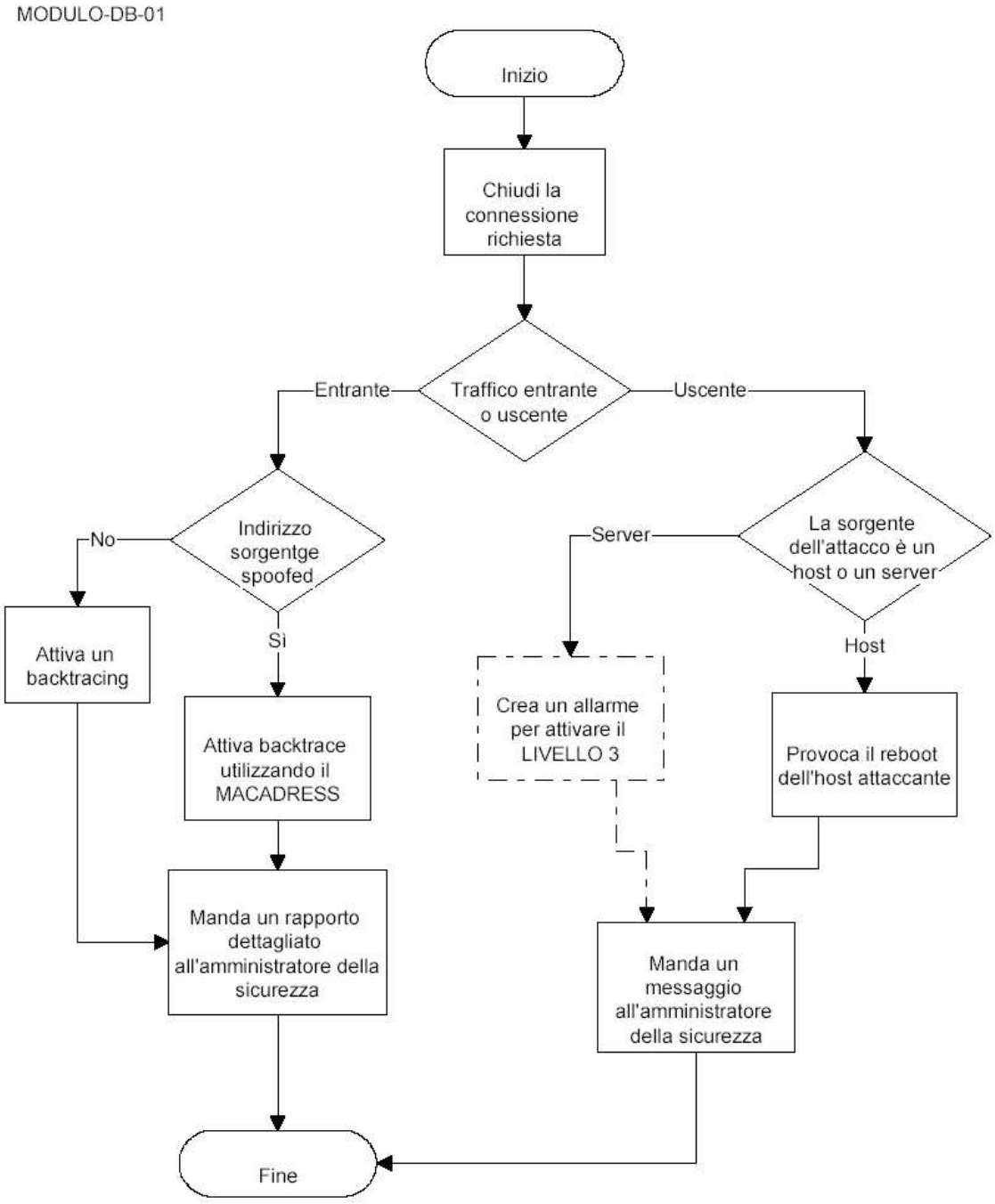

Figura 5.14: Albero decisionale del modulo DB 1

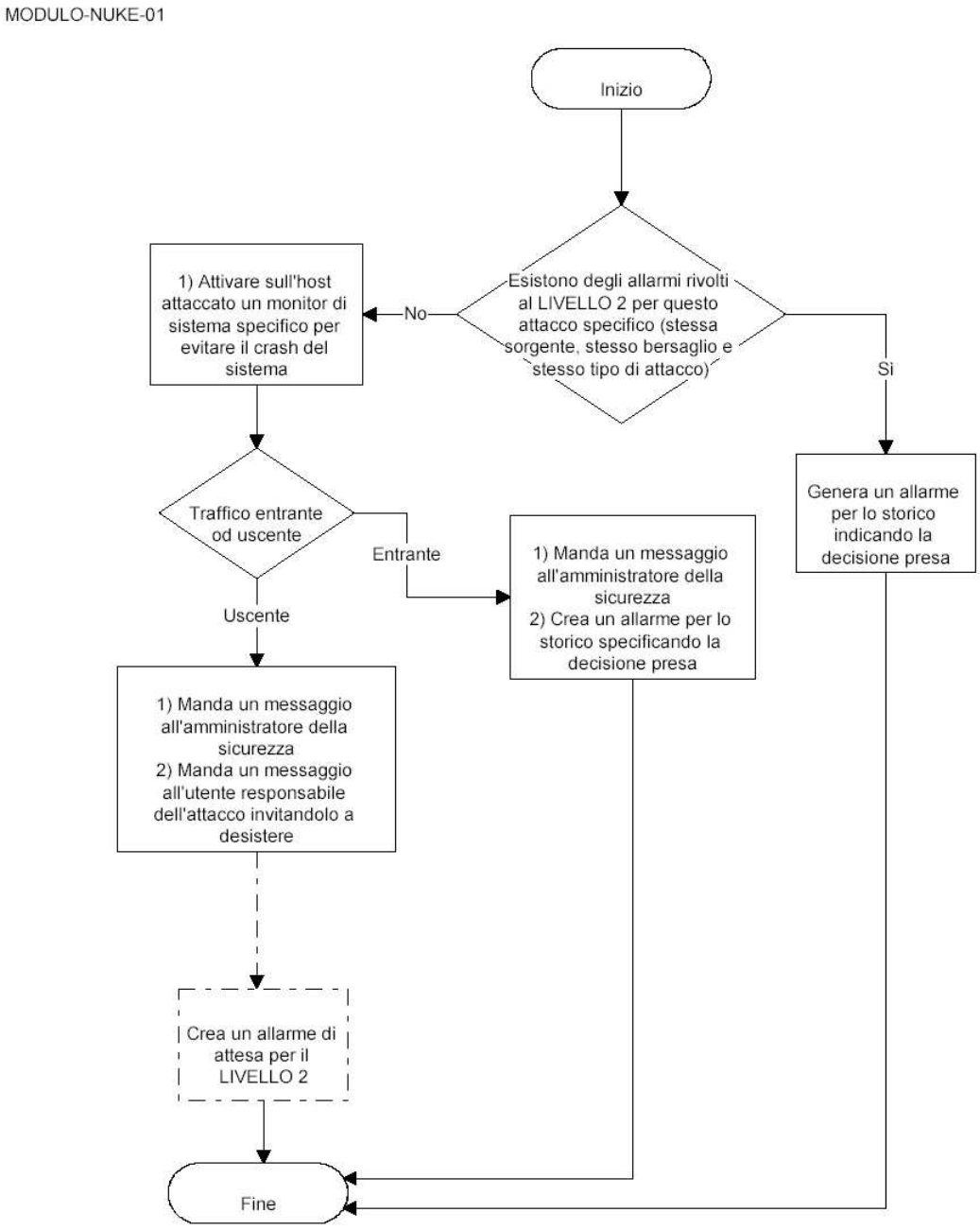

Tabella 5.3: Albero decisionale del modulo nuke 01

riguardo la struttura del Livello 2: l'attesa di X minuti che si attiva in questo livello è di tipo attivo, nel senso che viene periodicamente osservata la situazione degli allarmi generati dai sensori e non appena si evidenzia che l'attività illecita si ripresenta l'attesa termina e si attivano le contromisure del caso.

Se non esistono allari verso il Livello 02 significa che questo è il primo allarme di questo tipo registrato da Prelude IDS, quindi bisogna attivare un monitor sugli host interssati dall'attacco per evitare un crash del sistema stesso. Viene quindi analizzato l'allarme per scoprire se l'attacco è stato generato all'esterno della rete (traffico entrante) oppure al suo interno (traffico uscente). Nel primo caso la sola cosa che SLUNP può fare è avvertire l'utente tramite un rapporto dettagliato, mentre nel secondo caso possiede maggiore libertà d'azione: innanzi tutto avverte l'amministratore della sicurezza, quindi avverte l'utente di cessare la propria attività illecita, infine attiva l'attesa del Livello 02 grazie ad un allarme.

5.2.10 Modulo autentificazione root 01

Questo modulo, illustrato in figura 5.15, consente a SLUNP di riconoscere i login con diritti di superutente autorizzati e di contrastare quelli giudicati illegittimi.

Innanzitutto si verifica se il loging con diritti di superutente si verifica entro la finestra temporare prevista; se si riscontra una violazione si nega il permesso di login e si manda un rapporto dettagliato all'amministratore della sicurezza.

Nel primo caso viene attivato un tool per verificare l'autenticità dell'utente: degli intrusi potrebbero infatti riuscire a guadagnare diritti di superutente grazie a bug di sistema. Se il superutente è autentico si crea un allarme che Prelude Manager andrà a memorizzare all'interno del proprio database, altrimenti si forza il logout dell'utente che ha guadagnato in modo illecito i diritti di superutente e se ne revocano i diritti di login per impedirgli ulteriori accessi. Dopo aver mandato un rapporto all'amministratore della sicurezza si verifica se l'utente che ha effettuato il login illegittimo è un utente interno alla rete locale (utente locale) oppure esterno (utente remoto). In quest'ultimo caso, vista l'importanza del server (o dell'host) attaccato, il sistema esperto termina la sua esecuzione.

Se l'utente che ha tentato la connessione come superuser è interno alla rete locale, il sistema attiva delle procedure di backtracing per identificare il responsabile dell'attacco (infatti, la persona che ha tentato di penetrare le difese del sistema potrebbe aver utilizzato account fittizi per tentare di nascondere la sua vera identità);

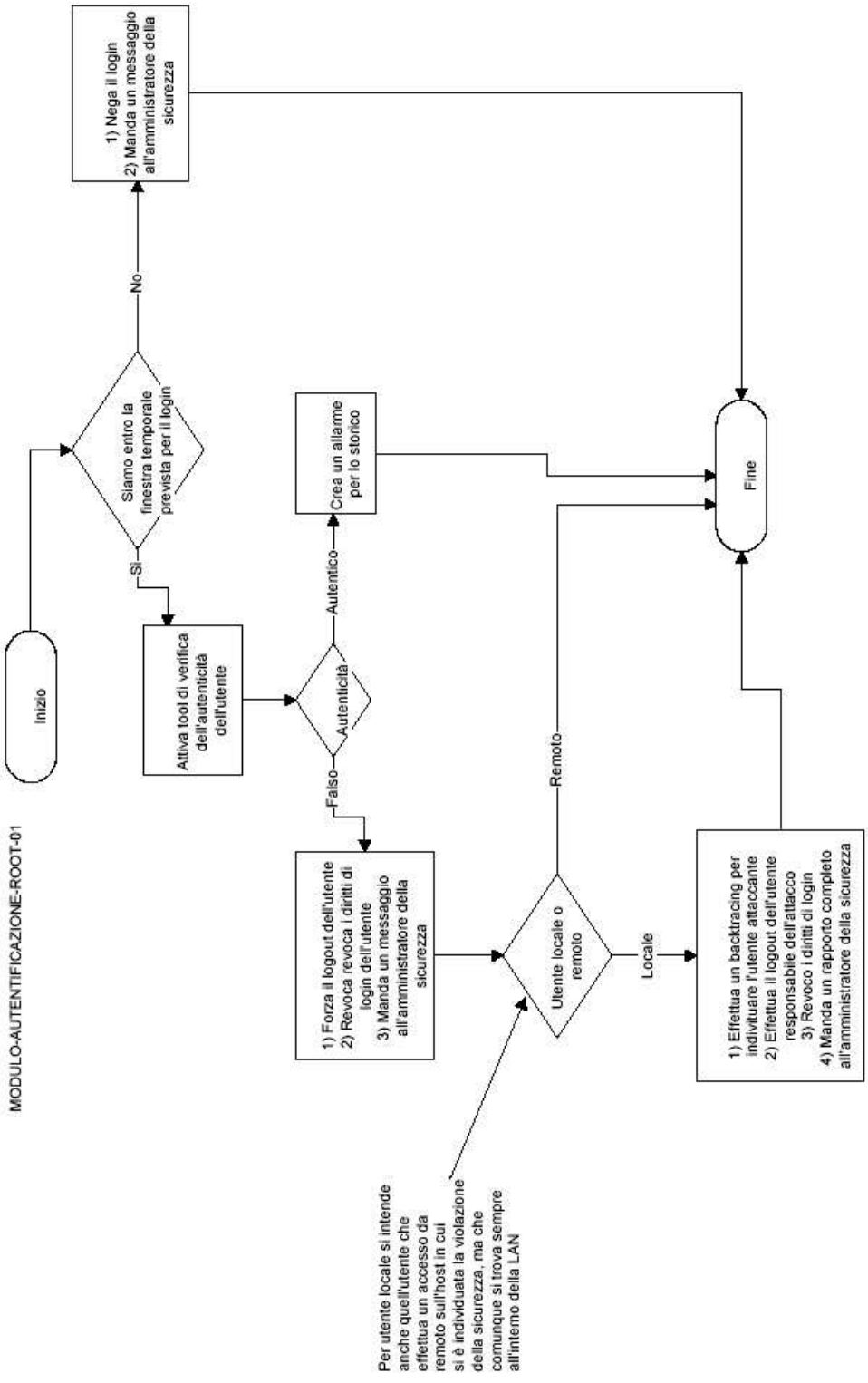

Figura 5.15: Albero decisionale del modulo autentificazione root 01

qualora venga identificato il responsabile dell'attività illecita se ne forza il logout dal sistema e si revocano i suoi diritti, quindi si manda un rapporto dettagliato all'amministratore della sicurezza.

5.2.11 Modulo worms 01

Come mostrato in figura 5.16, come prima cosa il sistema verifica che non vi siano allarmi rivolti al Livello 2 (si verifica la stessa situazione descritta nella sezione 5.2.9).

Se questi allarmi non esistono, significa che siamo in presenza del primo tentativo di attacco di questo tipo. Si procede quindi a verificare se la sorgente dell'attacco è interna alla lan (traffico uscente) oppure esterna alla rete locale (traffico entrante). Nel primo caso si manda un messaggio di avvertimento sia all'amministratore della sicurezza, sia all'utente a cui appartengono i processi worms attivi, quindi si attiva un monitor di sistema sull'host infettato dal worm con il compito di uccidere tutti i processi relativi. Viene quindi attivato il Livello 2 (vedi sezione 5.2.25) traminte un allarme di attesa.

Se la sorgente dell'attacco è esterna alla rete locale, si effettua un controllo per verificare il numero di connessioni presenti sulla porta interessata all'attacco. Se il loro numero risulta superiore al numero minimo di connessioni che si vuole comunque garantire, in qualsiasi circostanza la rete si trovi ad operare, il sitema attua una serie di contromisure per ridurre al livello minimo il numero di connessioni permesse, in modo da ridurre la velocità di propagazione "dell'infezione" creata dal worm. Vengono quindi avvertiti l'amministratore della sicurezza e tutti gli utenti della lan e viene anche attivata una serie di monitor di sistema che hanno il compito di uccidere tutti i processi worms presenti nella rete.

5.2.12 Modulo 03

L'albero decisionale in figura 5.17 viene seguito qualora all'interno del Livello 1 si riscontri che la rete locale ricopre un'importanza giudicata *media*.

Questo modulo presenta molte analogie con quello presentato in sezione 5.2.7; nelle sezioni successive vengono presentati solamente i moduli specifici.

5.2.13 Modulo RPC 01

L'albero decisionale di figura 5.18 mostra come il sistema SLUNP operi qualora ven-

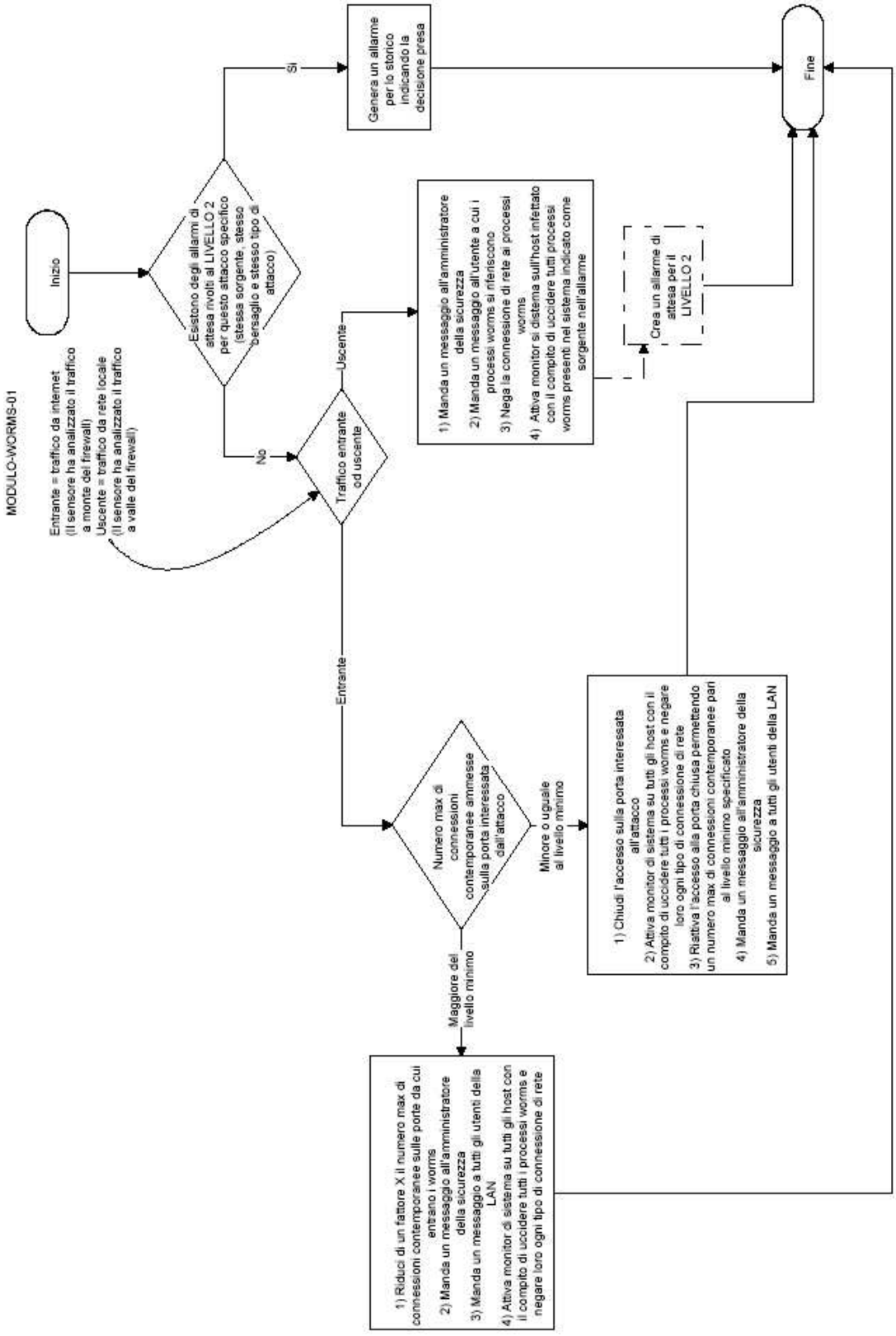

Figura 5.16: Albero decisionale del modulo worms 01

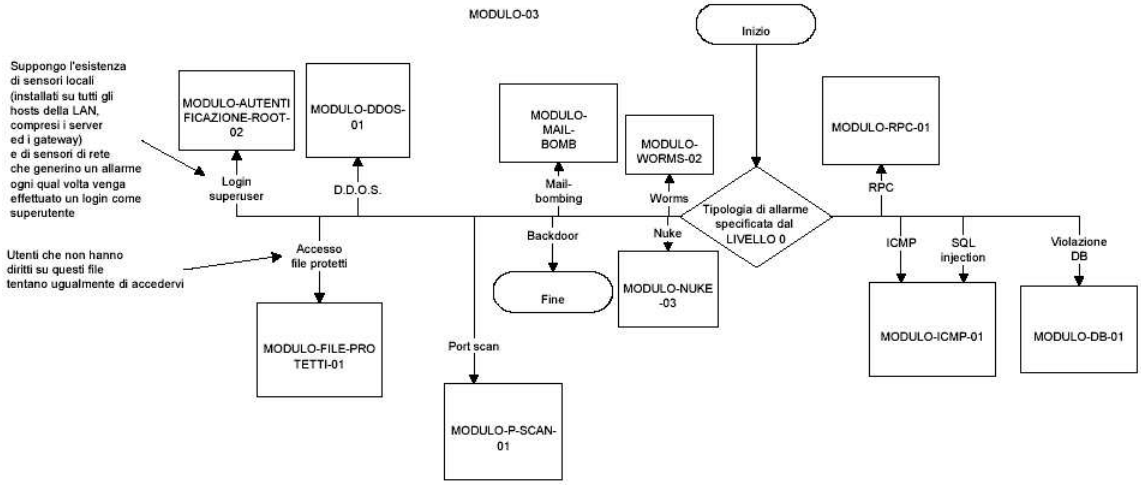

Figura 5.17: Albero decisionale del modulo 03

ga rilevato un attacco appartenente a questa tipologia. Innanzitutto viene chiusa la connessione stabilita (oppure viene rifiutata la richiesta di effettuare la connessione), quindi si stabilisce se la fonte dell'attacco si trovi all'interno della rete locale (traffico uscente) oppure al suo esterno (traffico entrante). Nel primo caso ci si limita a mandare un rapporto dettagliato all'amministratore della sicurezza, mentre nel secondo caso il rapporto viene mandato solamente dopo che si è determinato in maniera sicura la sorgente dell'attacco.

Per effettuare questa identificazione il sistema esperto verifica innanzitutto se l'indirizzo IP che il sensore ha specificato come sorgente dell'attacco è spoofed oppure no. Se non lo è si attiva un backtracing per raccogliere tutti i dati possibili sull'identità dell'attaccante; se l'indirizzo è stato alterato si attiva un backtracing sul mac address indicato dal sensore.

5.2.14 Modulo ICMP 01

In figura 5.19 viene mostrato l'albero decisionale seguito da SLUNP per fronteggiare un attacco che sfrutta le debolezze del protocollo ICMP. Come si può vedere, la prima contromisura adottata è quella di attivare su tutti gli host della rete dei tools che proteggano il sistema; viene quindi stabilito se la sorgente dell'attacco è interna alla rete (traffico uscente) oppure esterna (traffico entrante). Se la sorgente è interna si passa ad analizzare l'albero decisionale descritto nella sezione 5.2.15, altrimenti si

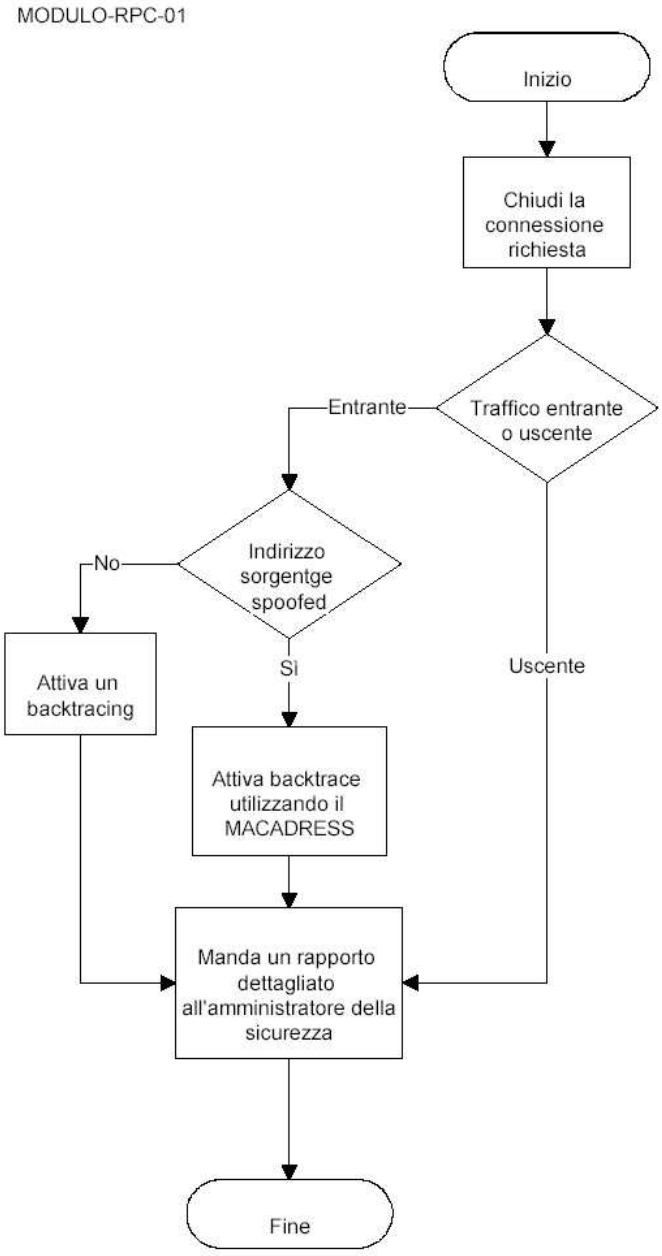

Figura 5.18: Albero decisionale del modulo RPC 01

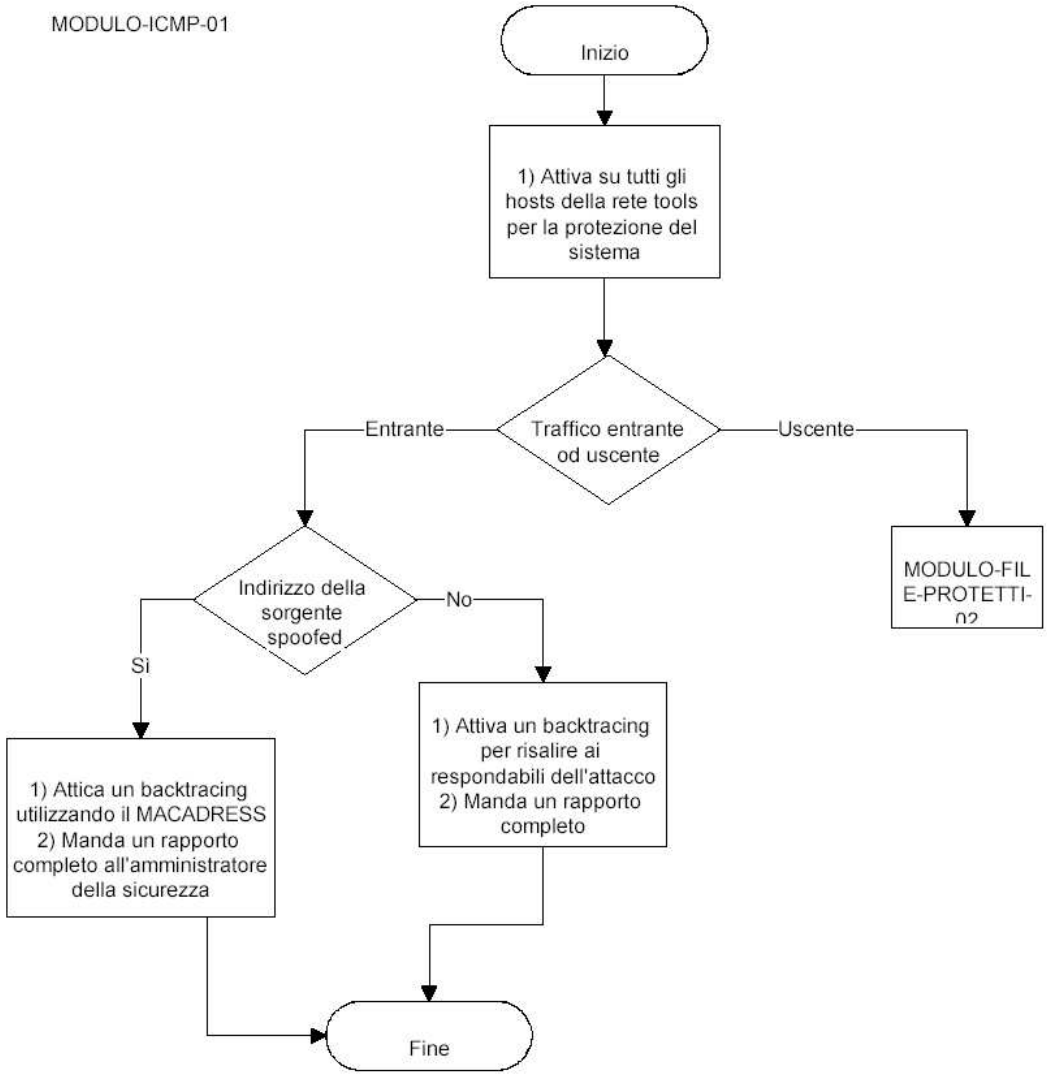

Figura 5.19: Albero decisionale del modulo ICMP 01

determina se l'indirizzo IP della sorgente è spoofed poiché questo dato è necessario per attivare un backtracing basato sull'indirizzo IP stesso (indirizzo non spoofed) oppure sul mac address specificato dal sensore (indirizzo spoofed). In ongi caso come ultimo passo prima della fine dell'albero decisionale si invia un rapporto dettagliato all'amministratore della sicurezza.

5.2.15 Modulo file protetti 02

La figura 5.20 mostra come vi siano due differenti procedimenti decisionali dipendenti dal fatto che la sorgente dell'attacco sia una workstation oppure un server.

Se la sorgente dell'attacco è una workstation si manda un messaggio all'amministratore della sicurezza e successivamente si forza il reboot del'host, quindi si crea un allarme che il manager andrà a memorizzare nel proprio databse. Sebbene la decisione di forzare il riavvio della macchina utilizzata per effettuare l'attacco sembri una reazione eccessiva (si pensi infatti ad un computer utilizzato da più utenti contemporaneamente che vedono persi i dati non salvati), essa risulta più che giustificata se si considera il livello di importanza della rete locale e dei dati in essa contenuti.

Se la sorgente dell'attacco è un server della rete locale si invia un rapporto dettagliato all'amministratore della sicurezza e si induce il logout dell'utente responsabile dell'attacco, quindi si identificano i diritti dell'utente stesso. Se si tratta di un semplice utente allora gli vengono revocati tutti i diritti di login e si invia un rapporto all'amministratore della sicurezza; se si tratta del superutente si termina l'esecuzione dell'albero decisionale.

5.2.16 Modulo nuke 03

Il primo controllo che viene eseguito nello schema di figura 5.21 è quello relativo alla presenza di allarmi rivolti al Livello 2 (vedi sezione 5.2.25) relativi allo stesso tipo di attacco, stessa sorgente e stessa destinazione; se ve ne sono, si crea un allarme che Prelude Manager andrà a memorizzare nel proprio database e quindi si termina l'esecuzione di questo albero decisionale.

Se non esistono allarmi rivolti al Livello 2 significa che questo è il primo allarte ricevuto per questa tipologia di attacco, quindi si procede attivando sull'host attaccato un tool in grado di evitare il crash del sistema, quindi si procede con il de-

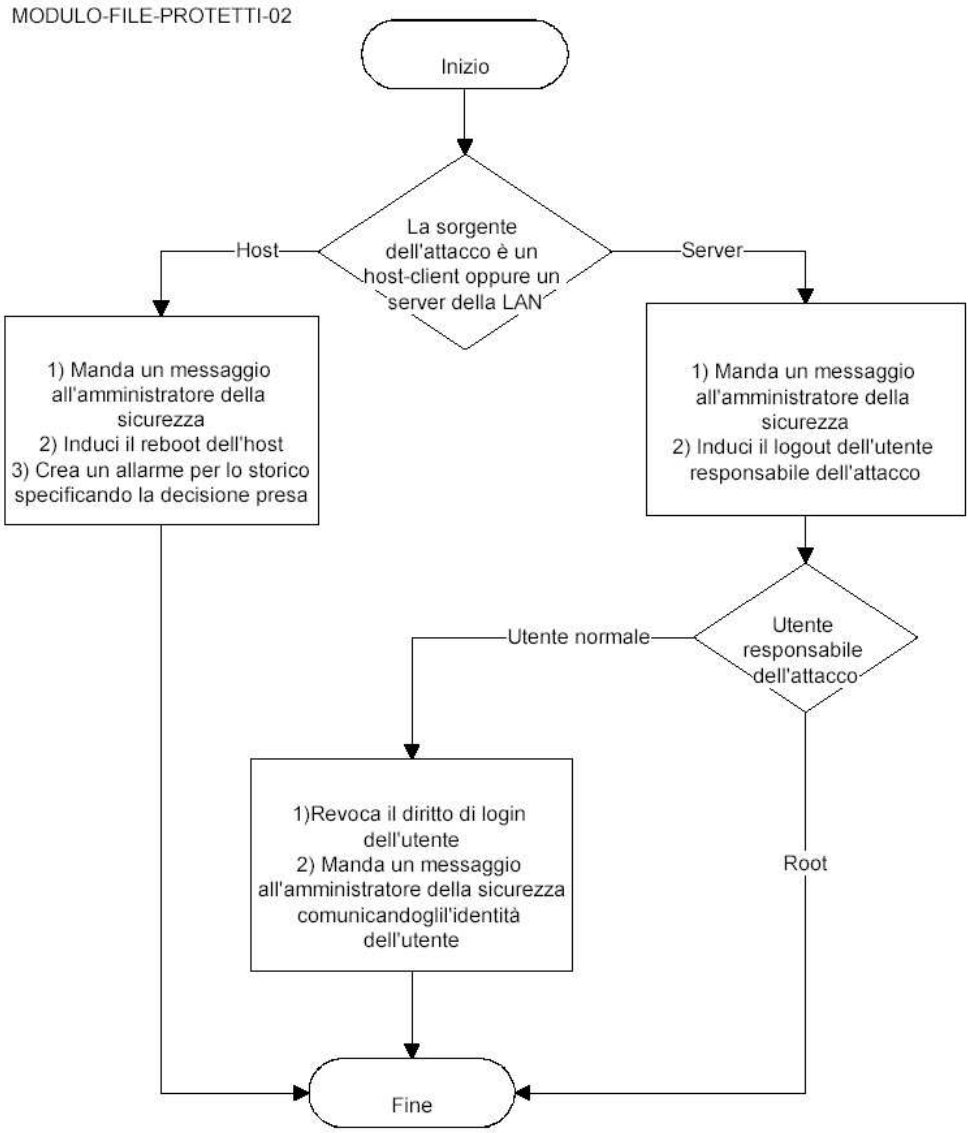

Figura 5.20: Albero decisionale del modulo file protetti 02

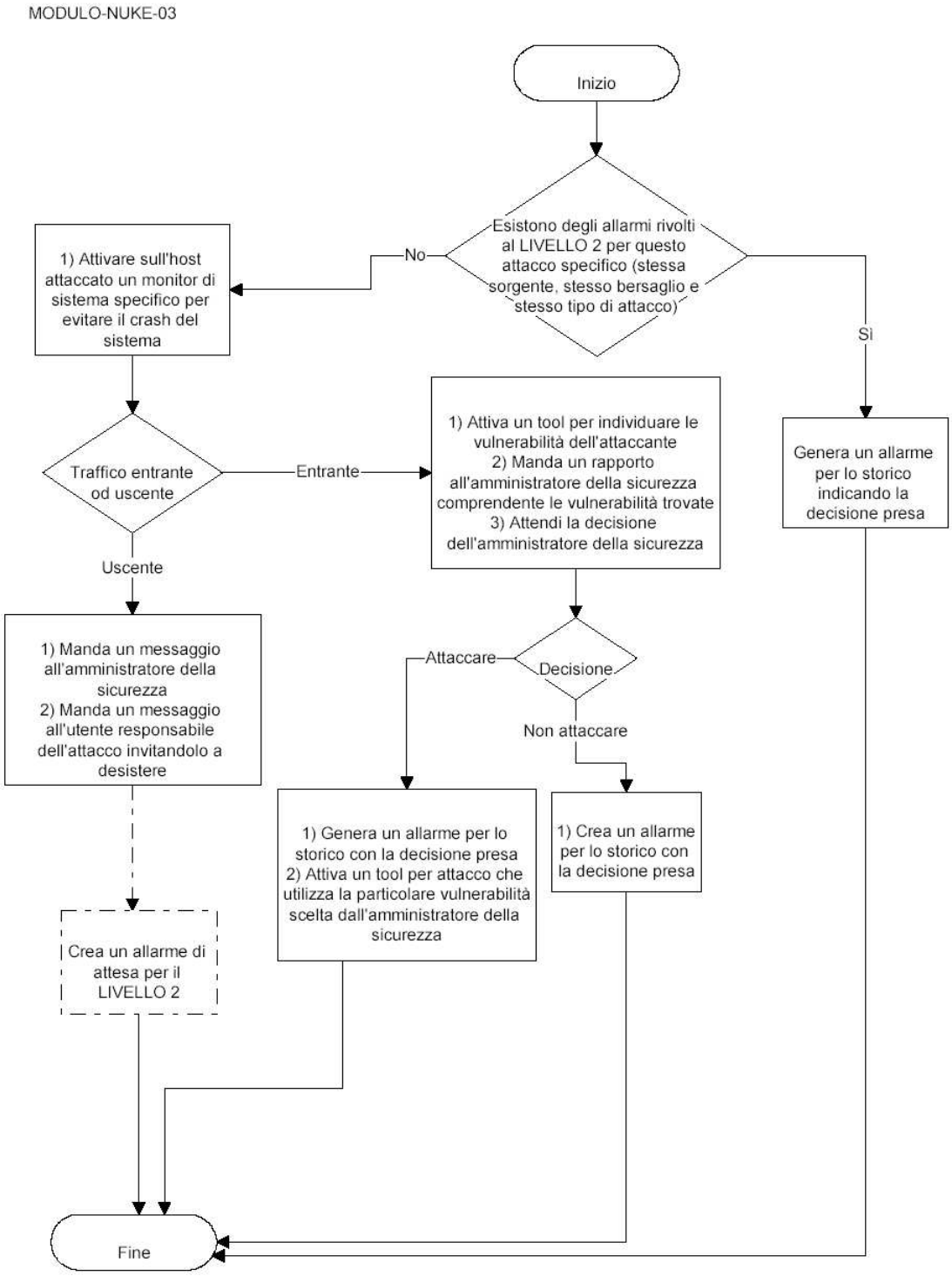

Figura 5.21: Albero decisionale del modulo nuke 03

terminare se l'attacco proviene dall'interno della rete locale (traffico uscente) oppure dal suo esterno (traffico entrante). Se la sorgente dell'attacco risulta essere interna alla lan si procede mandando un messaggio all'amministratore della sicurezza ed un messaggio di avvertimento all'utente responsabile dell'attacco, quindi si crea un allarme rivolto al Livello 2.

Se la sorgente dell'attacco risulta essere esterna alla rete locale, data l'importanza della rete che deve proteggere, il sistema SLUNP procede ad attivare contromisure di tipo attivo contro la sorgente dell'attacco, anche se questa non si trova sotto la giurisdizione dell'amministratore della sicurezza locale. SLUNP procede tentando di individuare tutte le vulnerabilità della sorgente dell'attacco, quindi effettua un rapporto dettagliato specificando la situazione della rete locale e tutte le vulnerabilità trovate. Attende quindi che l'amministratore della sicurezza a prenda la decisione finale: deve essere ben conscio delle ripercussioni legali nel caso decisa di far proseguire la reazione attiva nei confronti dell'attaccante e deve possedere speciali autorizzazioni fornitegli dai responsabili legali. La decisione di attivare le contromisure attive deve essere fortemente giustificata dai danni che potenzialmente il successo dell'attacco sarebbe in grado di provocare e dall'importanza della rete che si stà proteggendo.

Qualora si decida di proseguire sulla strada suggerita da SLUNP, l'amministratore seleziona una o più contromisure elencate nell'interfaccia real-time del sistema esperto il quale le attiverà e terminerà l'esecuzione di questo albero decisionale.

In caso contrario, SLUNP creerà un allarme che verrà memorizzato nel database di Prelude Manager.

5.2.17 Modulo pscan 01

Come mostrato in figura 5.22, si verifica che non esistano allarmi rivolti al Livello 2 (vedi sezione 5.2.25), dopo di ché si procede all'attivazione su tutti gli host della rete locale tools di simulazione di vulnerabilità in modo da ingannare il port scan in corso, quindi si avverte l'amministratore della sicurezza inviandogli un rapporto sulla situazione attuale della lan, quindi si monitora il flusso di dati provenienti dalla sorgente dell'attacco e si attiva il Livello 2.

Se non vi sono allarmi rivolti al Livello 2, si crea un allarme per lo storico del database e si termina l'esecuzione dell'albero decisionale.

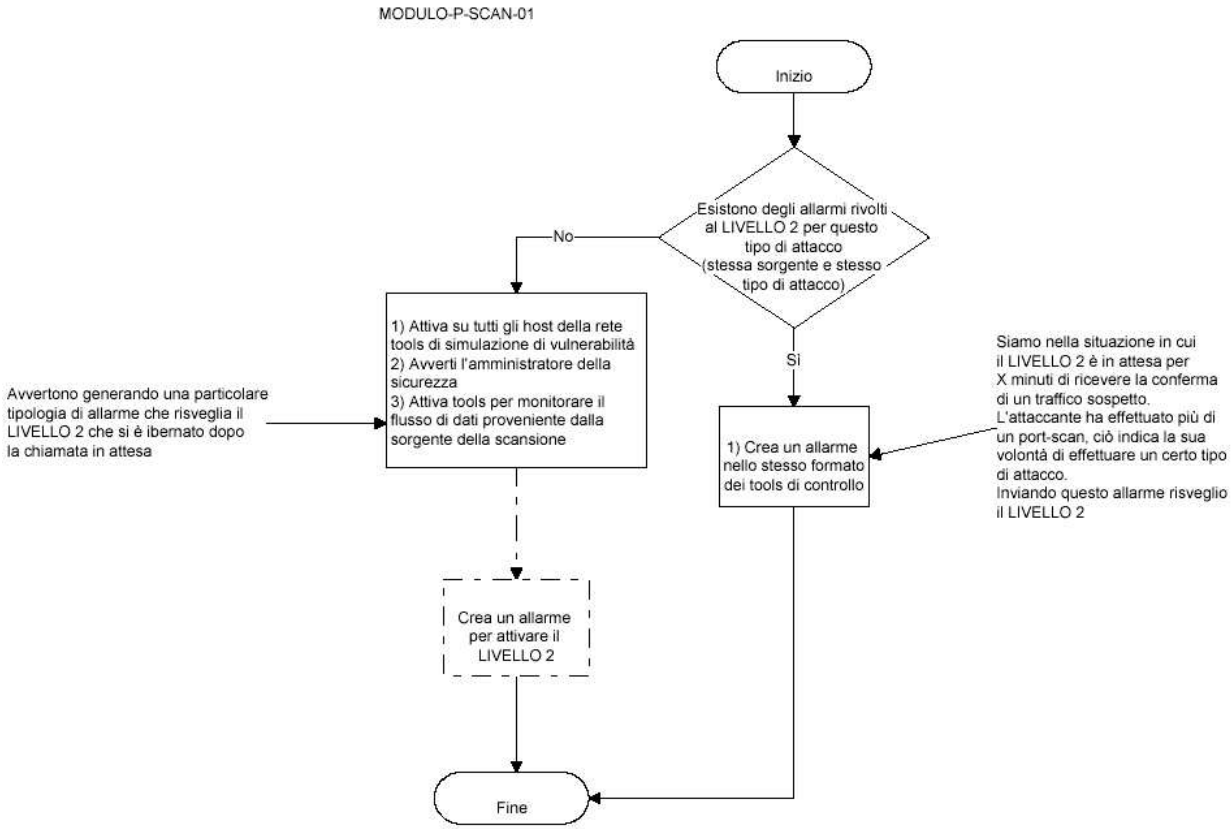

Figura 5.22: Albero decisionale del modulo pscan 01

5.2.18 Modulo accesso file protetti 01

L'albero decisionale che è mostrato in figura 5.23 viene seguito dopo che si è ottenuto l'accesso ai file oggetto della protezione. Se non esistono allarmi rivolti al Livello2, SLUNP provvede a negare l'accesso ai file protetti e ad attivare tools di protezione per impedire ulteriori accessi non autorizzati ai file. Tramite l'utilizzo di backup sicuri vengono reintegrati i file compromessi e quindi viene avvertito l'amministratore della sicurezza tramite un rapporto dettagliato. Viene quindi effettuata una verifica per determinare se l'utente sia connesso sullo stesso host in cui si è verificata la violazione della sicurezza oppure se vi sia connesso tramite accesso remoto.

Nel primo caso si verifica se l'utente è legittimo oppure no. In caso affermativo lo si avvisa che i file modificati sono protetti dal sistema e si attiva il Livello 2 (vedi sezione 5.2.25) tramite la creazione di un allarme appropriato, altrimenti si adottano altri tipi di contromisure tra cui la revoca totale dei diritti di login dell'utente nel sistema.

5.2.19 Modulo autentificazione root 02

Apparentemente l'albero decisionale mostrato in figura 5.24 non è di facile lettura, ma una più attenta analisi lo renderà comprensibile. Innanzitutto si verifica che il login sia stato effettuato entro la finestra temporale prevista dall'amministratore della sicurezza. All'interno della finetra si attiva un controllo sull'autenticità dell superutente. Se è un utente autorizzato si crea un semplice allarme per lo storico, altrimenti si forza il logout dell'utente e se ne revocano i diritti all'interno del sistema, quindi si manda un rapporto dettagliato all'amministratore della sicurezza. Se il login è stato effettuato al di fuori della finestra temporale prevista si nega il permesso di accesso e si effettua un rapporto all'amministratore della sicureza.

Sia nel caso che si tenti di effettuare il login al di fuori della finestra temporale, sia che l'utente che lo ha effettuato entro i termini previsti sia un utente fittizio, si procede a stabilire se l'utente sia locale oppure remoto. Nel caso in cui l'utente risulti essere interno si effettua un backtracing per identificarlo, quindi di forza il suo logout e se ne revocano i diritti di login, quindi si manda un rapporto dettagliato all'amministratore della sicurezza. Se al contrario l'utente risulti essere remoto, si effettua un backtracing per tentare di identificarlo (utilizzando il suo indirizzo IP o il suo mac address nel caso in cui l'indirizzo IP risulti essere spoofed).

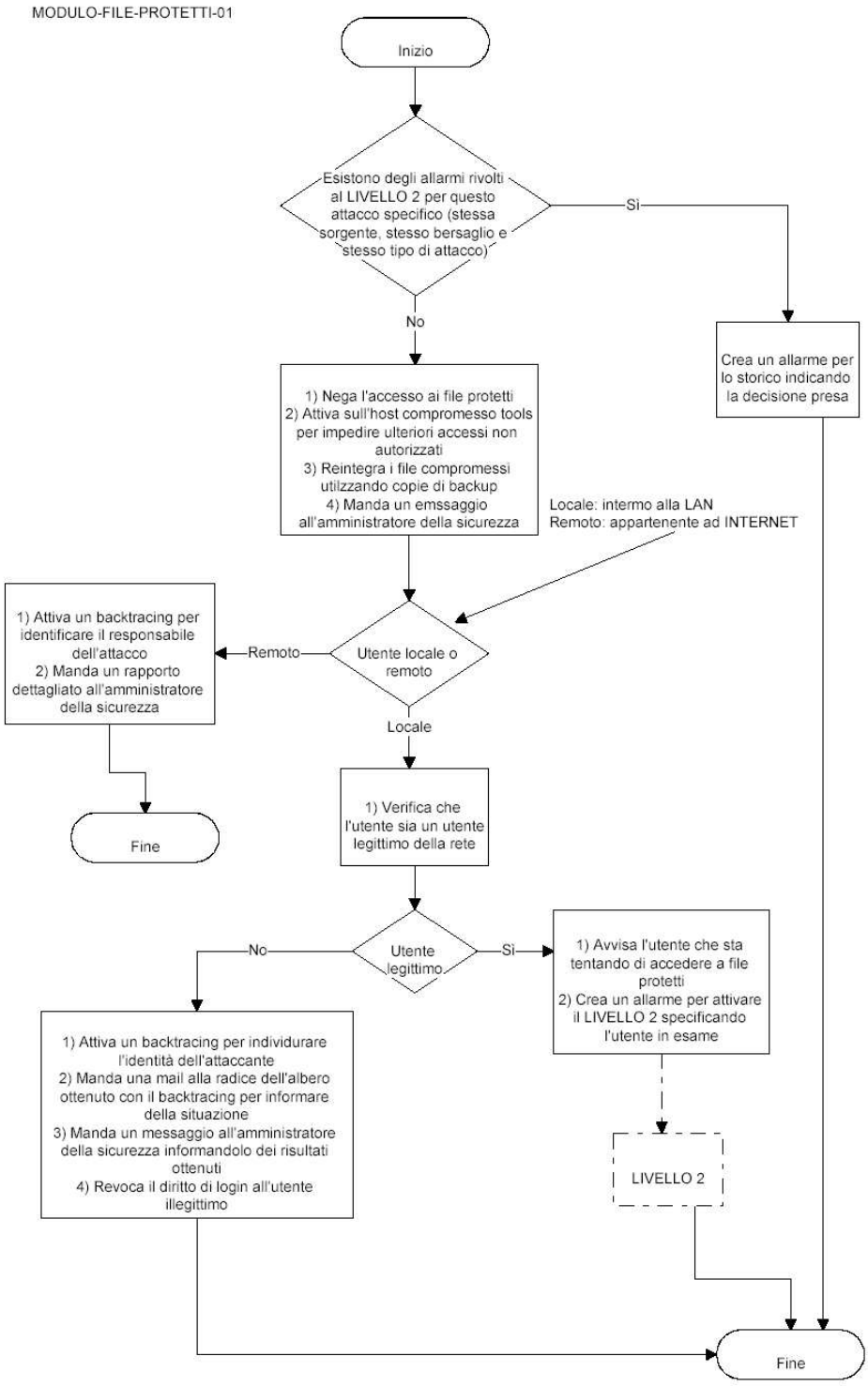

Figura 5.23: Albero decisionale modulo file protetti 01

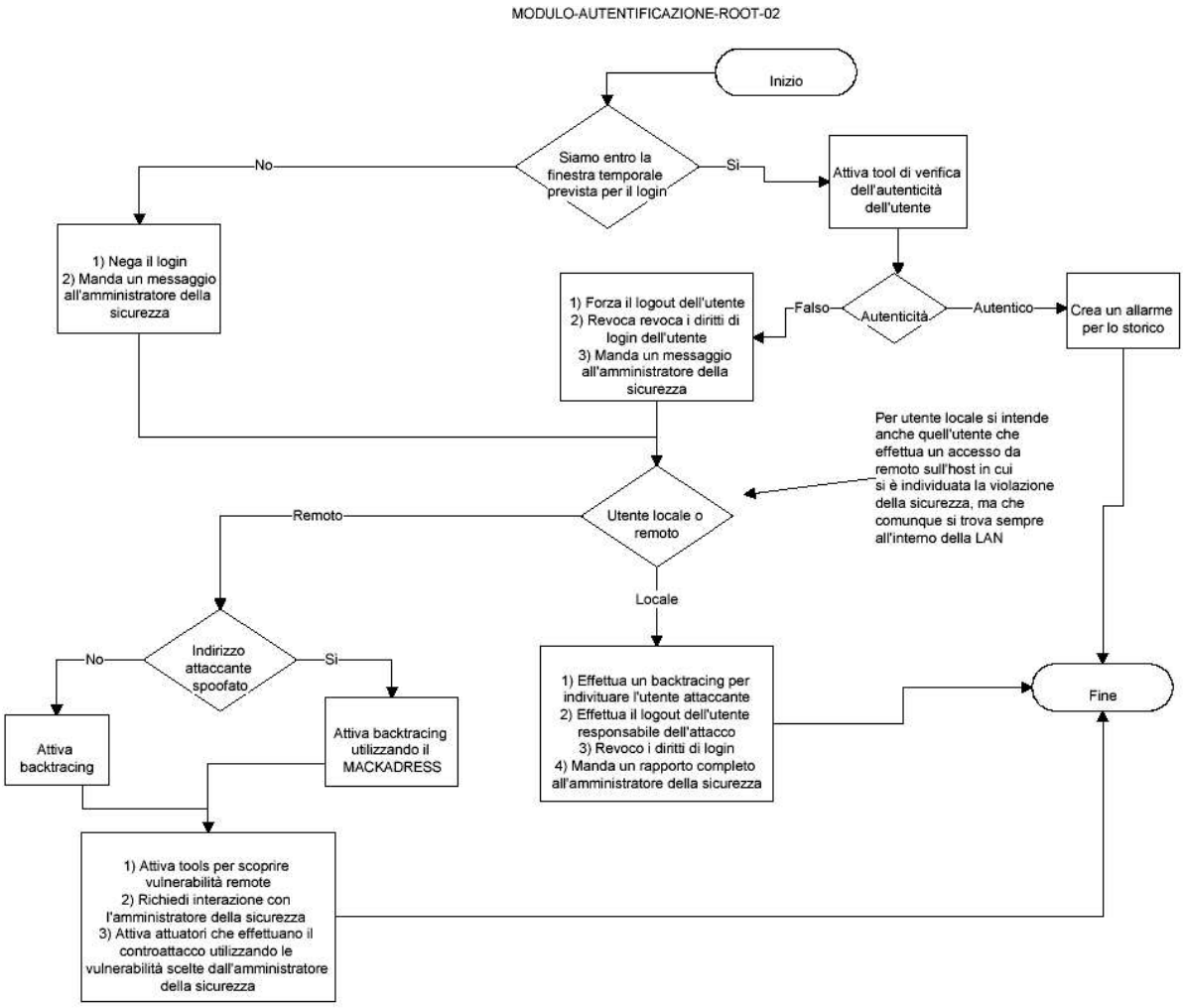

Figura 5.24: Albero decisionale del modulo autentificazione root 02

Figura 5.25: Albero decisionale del modulo D.D.o.S. 02

Viene quindi attivato un tool di analisi delle vulnerabilità dell'attaccante, quindi procede come descritto nella sezione 5.2.16, inviando all'amministratore della sicurezza un rapporto dettagliato sulla situazione corrente della rete e sulle vulnerabilità scoperte; si attende la decisione dell'amministratore che può scegliere di procedere con una determinata contromisura oppure non attivarne nessuna.

5.2.20 Modulo D.D.o.S. 02

Come mostrato in figura 5.25, questo albero decisionale richiama quello descritto nella sezione 5.2.21 e prevede poi la creazione di un allarme per il Livello 2 (vedi sezione 5.2.25).

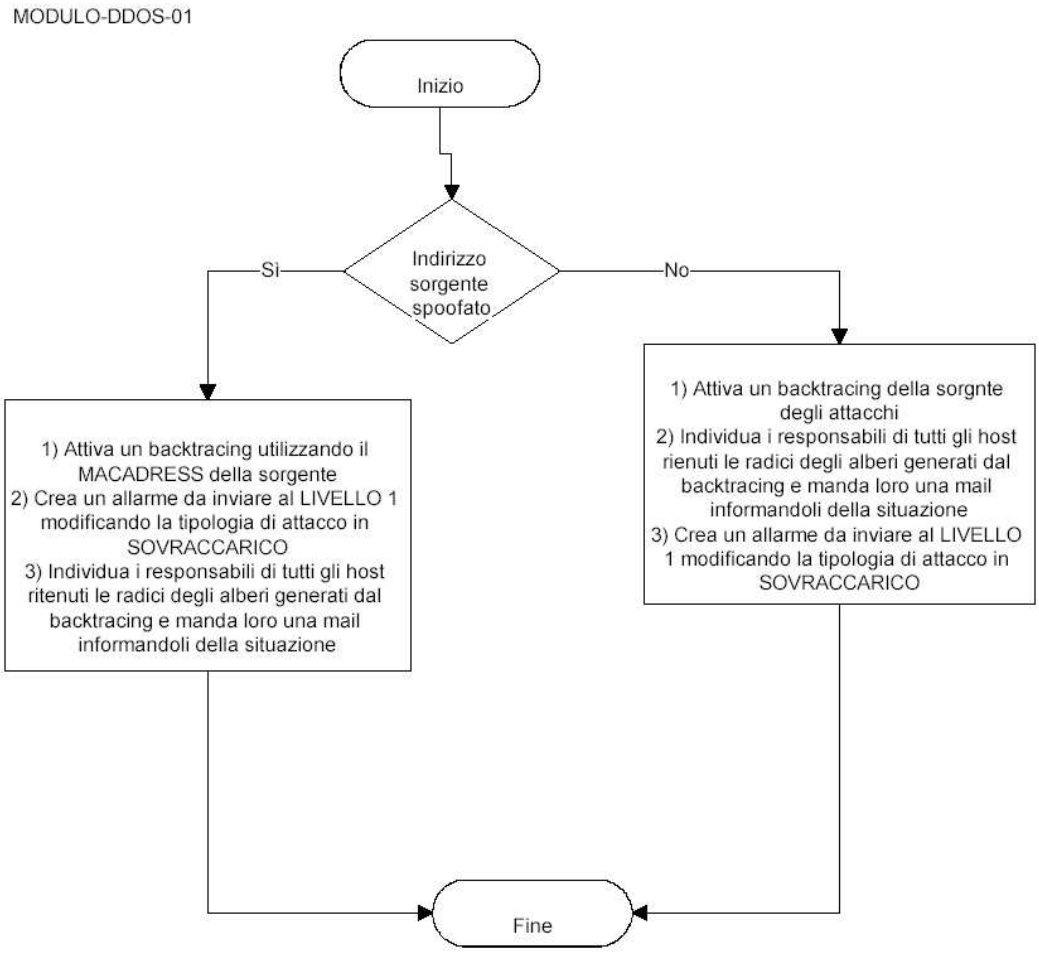

Figura 5.26: Albero decisionale del modulo D.D.o.S. 01

5.2.21 Modulo D.D.o.S. 01

Vista la natura distribuita dell'attacco, l'albero decisionale di figura 5.26 prevede un semplice controllo per verificare se l'indirizzo IP della sorgete dell'attacco sia spoofed o meno. Tale controllo viene utilizzato poiché le contromisure relative a questo tipo di attacco prevedono l'attivazione di un backtracing per ricostruire tutto l'albero di nodi utilizzati per l'attacco. E' quindi necessaro stabilire se si possa utilizzare l'indirizzo IP fornito dal sensore oppure l'indirizzo mac address; terminato il backtracing si crea un allarme per il Livello 1 (vedi sezione 5.2.4) indicando che nella rete è presente un sovraccarico, quindi vengono mandate delle e-mail a tutti i responsabili dei nodi che vengono utilizzati per l'attacco distribuito.

5.2.22 Modulo worms 02

Come mostrato in figura 5.27 (ed in maggior dettaglio in figura 5.28 e figura 5.29) come prima cosa il sistema SLUNP verifica che non vi siano allarmi rivolti al Livello 2 (vedi sezione 5.2.25) per lo stesso tipo di allarme, stessa sorgente e stessa destinazione. Se esistono si crea un semplice allarme per lo storico del database; se non esistono significa che siamo di fronte al primo allarme relativo a questa tipologia di attacco e quindi bisogna attivare le contromisure più adeguate al caso.

Si determina se la sorgente dell'attacco è interna alla rete (traffico uscente) oppure esterna (traffico entrante). Se la sorgente è interna si mandano due messaggi di avviso (uno rivolto all'amministratore della sicurezza e l'altro rivolto all'utente a cui i processi illegali appartengono), quindi si attiva un monitor sull'host infettato al fine di limitare il più possibile il dilagare dei worms. Viene quindi attivato il Livello 2.

Se la sorgente dell'attacco è esterna, il percorso decisionale è simile a quello presentato nella sezione 5.2.11, con la differenza che in questo caso l'importanza della rete è maggiore rispetto a quella presa in considerazione in tale sezione e quindi il sitema è autorizzato ad effettuare una ricerca delle vulnerabilità dell'attaccante ed a presentarle in un rapporto dettagliato che comprende anche la situazione attuale della rete. Starà all'amministratore della sicurezza decidere se permettere al sistema di attivare delle contromisure attive oppure no.

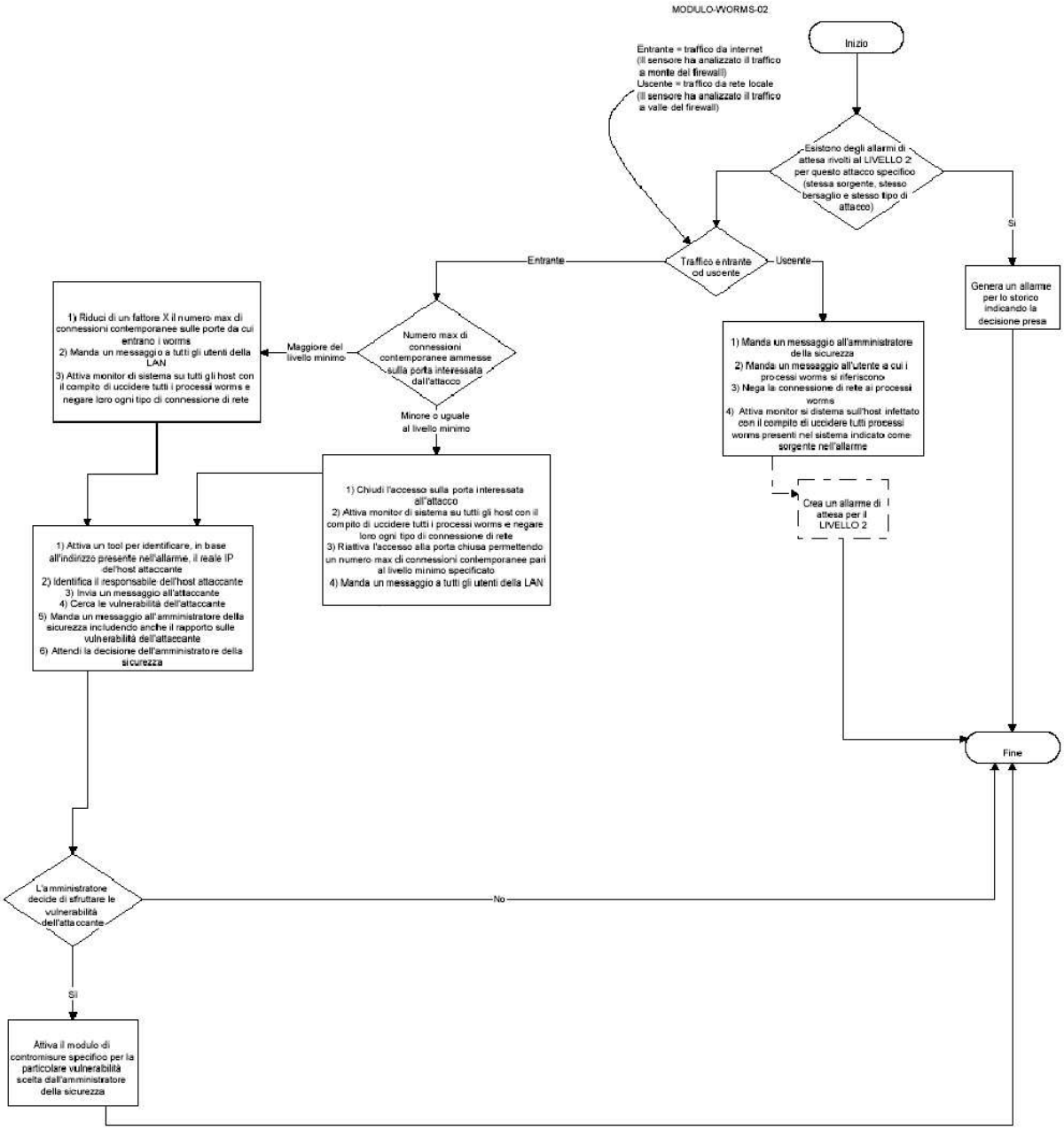

Figura 5.27: Albero decisionale del modulo worms 02

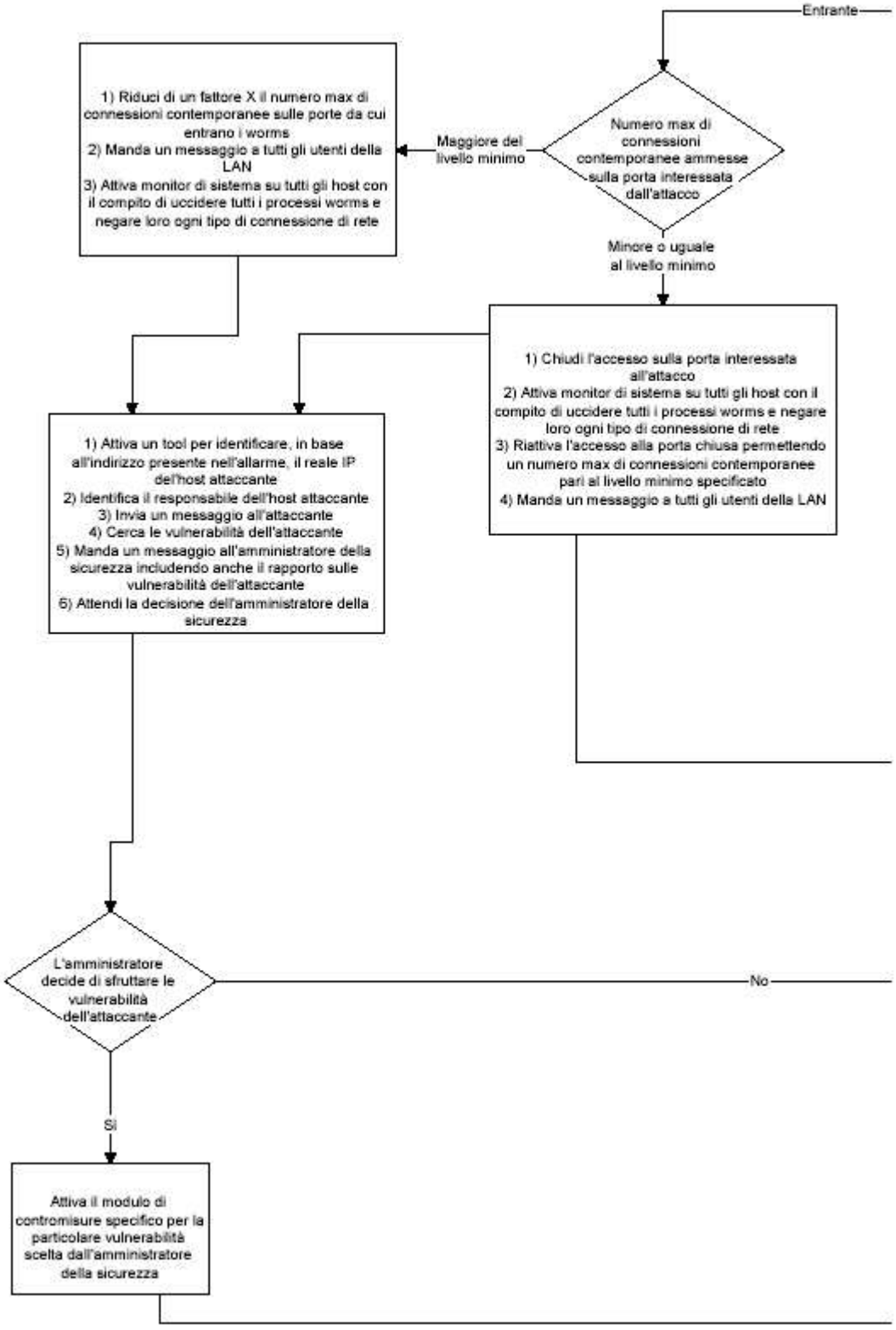

Figura 5.28: Albero decisionale del modulo worms 02 (prima parte)

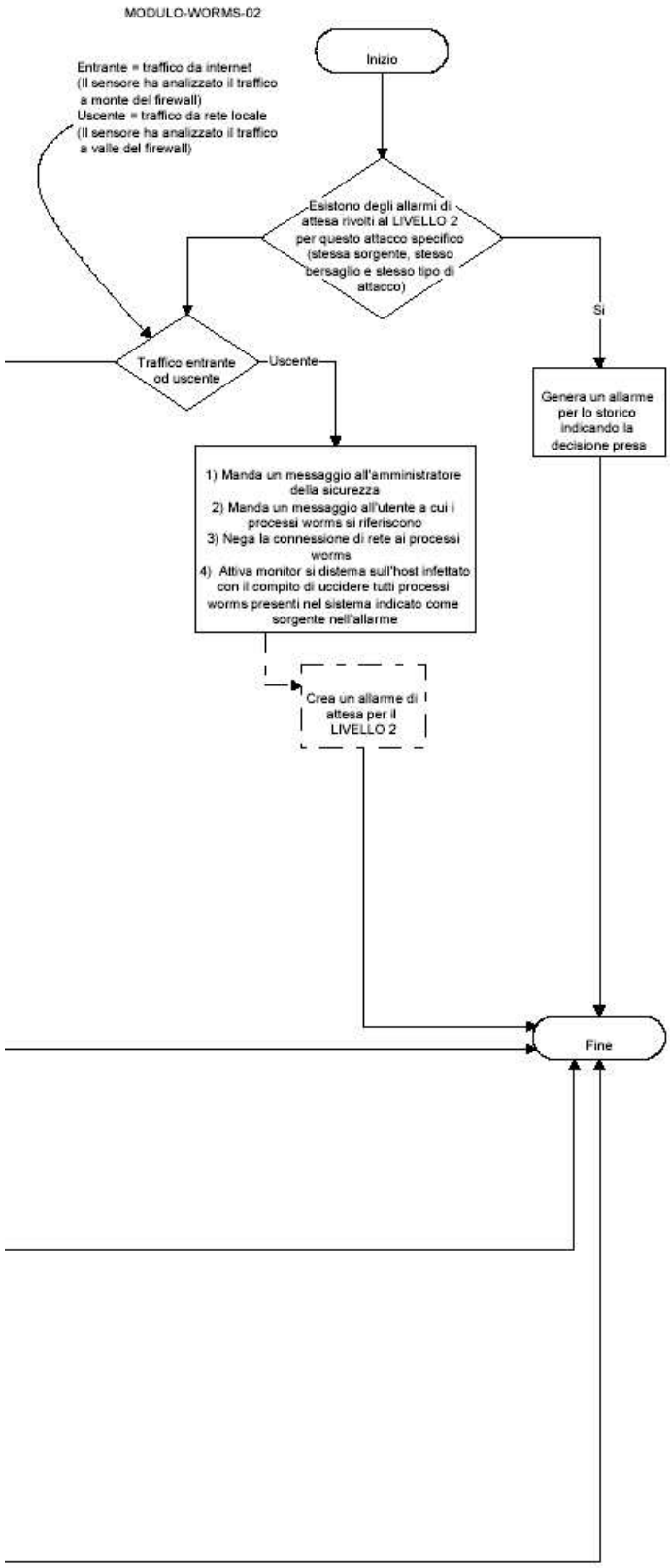

Figura 5.29: Albero decisionale del modulo worms 02 (seconda parte)

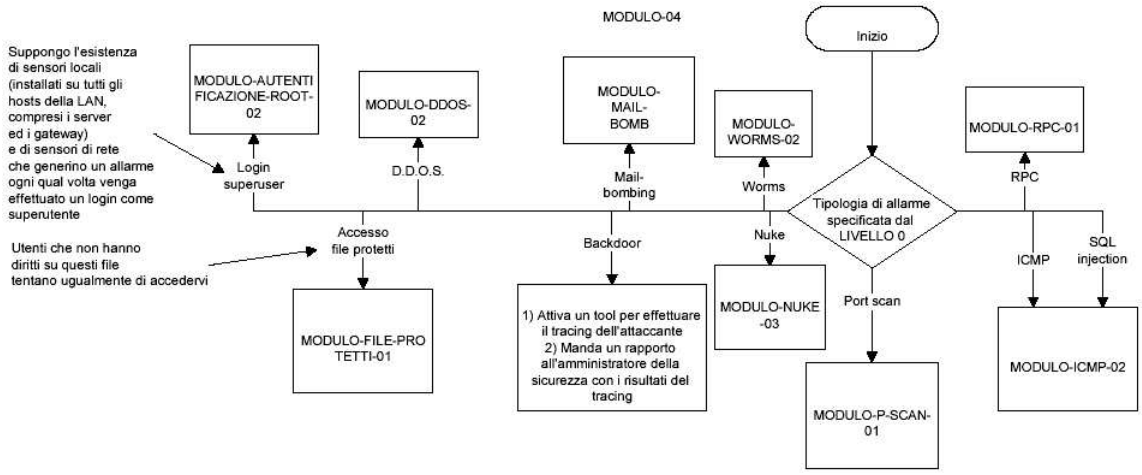

Figura 5.30: Albero decisionale del modulo 04

5.2.23 Modulo 04

Questo modulo, mostrato in figura 5.30, rappresenta il percorso decisionale che viene seguito da SLUNP qualora l'importanza della rete che dobbiamo proteggere venga giudicata di livello alto (vedi sezione 5.1). Il Modulo 04 presenta molte analogie con i moduli descritti in sezione 5.2.7 ed in sezione 5.2.12, quindi nelle sezioni successive presenteremo solamente i moduli decisionali specifici per attivare le contromisure atte alla difesa di una rete ad alta importanza.

5.2.24 Modulo ICMP 02

Questo modulo, rappresentato in figura 5.31, prevede l'attuazione di una prima serie di contromisure consistenti in tools capaci di proteggere gli host della rete dall'attacco in atto. In un secondo momento si effettua una verifica per stabilire se l'attacco provenga dall'interno della rete (traffico uscente) oppure dal suo esterno (traffico entrante). Nel primo caso si seguie l'albero decisionale del Modulo file protetti 02 (vedi sezione 5.2.15), mentre nel secondo caso si procede alla determinazione dell'autenticità dell'indirizzo IP che il sensore ha identificato come sorgente dell'attacco. Questa verifica viene utilizzata dal sistema esperto SLUNP per attivare un backtracing utilizzando l'indirizzo IP stesso (indirizzo non spoofed) oppure il mac address fornito dal sensore (indirizzo IP della sorgente spoofed). Indipendentemente dal risultato della verifica sull'indirizzo IP viene mandato un rapporto all'amministratore

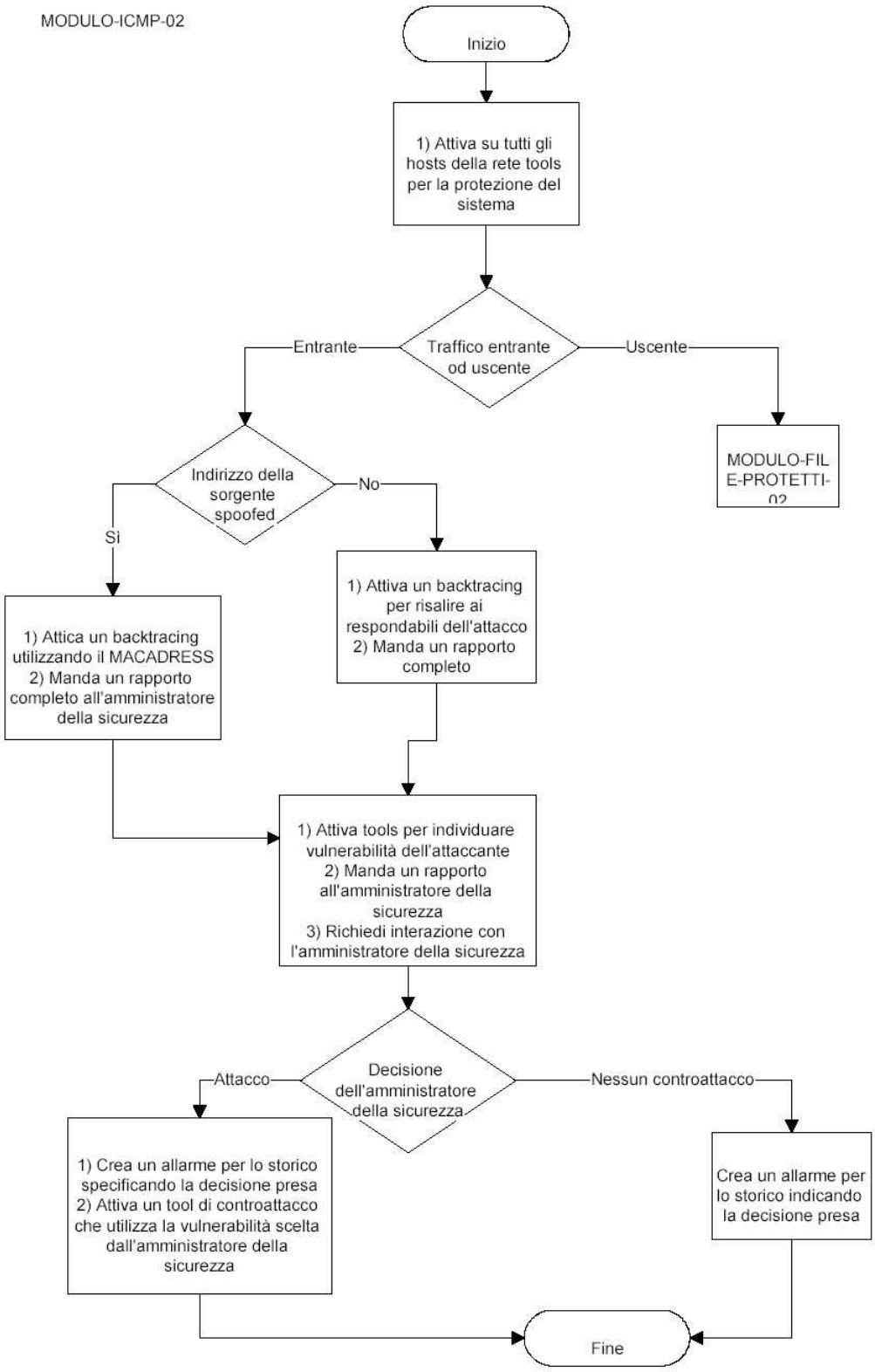

Figura 5.31: Albero decisionale del modulo ICMP 02

della sicurezza, quindi si attiva un tool di analisi delle vulnerabilità del sistema che il backtracing ha identificato come vera sorgente dell'attacco. Si manda un rapporto dettagliato indicando la situazione attuale della rete locale e le vulnerabilità trovate, nonché una serie di possibili reazioni che sfruttano tali vulnerabilità. Viene quindi lasciata all'amministratore della sicurezza la scelta di procedere con la reazione oppure bloccare l'attivazione delle contromisure attive. Se si decide di procedere vengono attivati i tool selezionati dall'amministratore della sicurezza, quindi si crea un allarme che verrà inserito all'interno del database di Prelude Manager; questo allarme verrà creato anche nel caso in cui si decida di non procedere oltre.

5.2.25 Livello 2: quarto livello decisionale di SLUNP

In figura 5.32 viene mostrato come questo livello decisionale venga attivato dagli allarmi provenienti dal Livello 01. Più precisamente, tali allarmi possono appartenere solamente a due tipologie: possono riguardare un attacco di tipo *port scan* oppure essere allarmi di *attesa*. Nel primo caso viene seguito l'albero decisionale che rappresenta il modulo pscan 02, nel secondo caso si attiva una verifica dipendente dal diverso tipo di attacco che si vuole fronteggiare: infatti, il Livello 01 non si liminta ad indicare al Livello 02 di mettersi in attesa, ma specifica anche il tipo di attacco che è stato rilevato originariamente dal sensore. Questo permette al Livello 02 di verificare, traminte strumenti specifici per ogni allarme, se entro un determinato periodo di tempo (X minuti o Y secondi) specificato dall'amministratore della sicurezza, non si verifichino altri tentativi di attacco. Se ciò non avviene significa che la sorgente dell'attacco ha desistito dai suoi intenti oppure che le contromisure adottate nei livelli decisionali inferiori hanno avuto effetto; se si riscontrano altri tentativi di attacco significa che la sorgente ha dichiarate intenzioni ostili nei confronti della rete locale e quindi è necessario prendere ulteriori provvedimenti di difesa. Vengono quindi seguiti alberi decisionali specifici per ogni tipologia di attacco originariamente rilevata dal sensore che tengono conto di quanto è stato fatto nei livelli decisionali precedenti e delle dichiarate intenzioni ostili della sorgente dell'attacco in corso.

5.2.26 Modulo nuke 02

Questo modulo, il cui albero decisionale è rappresentato in figura 5.33, prevede l'attuazione di una verifica per stabilire se la sorgente dell'attacco è un semplice

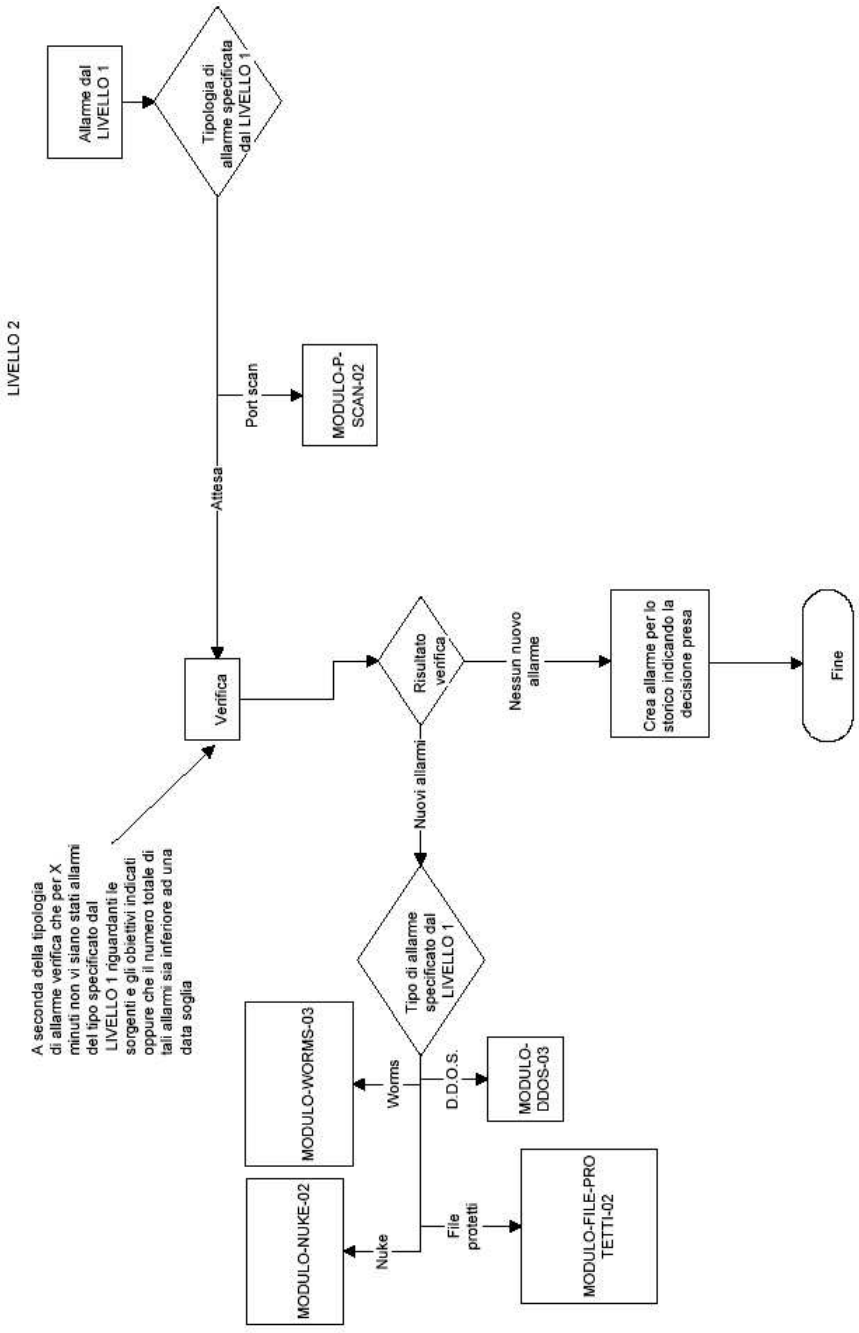

Figura 5.32: Albero decisionale del livello 02

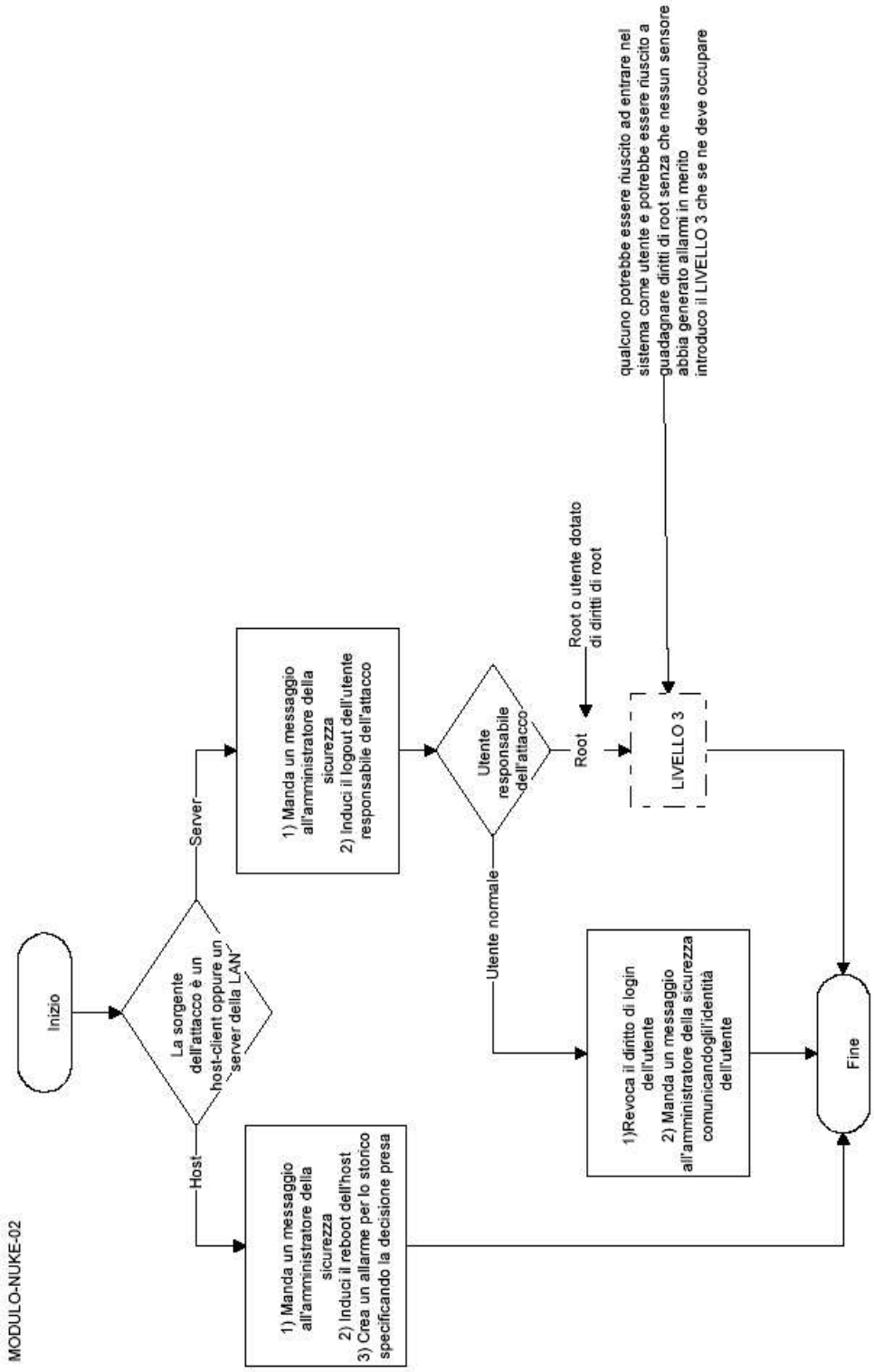

Figura 5.33: Albero decisionale del modulo nuke 02

host oppure un server della rete locale. Nel primo caso si manda un messaggio all'amministratore della sicurezza e si induce il reboot dell'host, quindi viene creato un allarme che verrà memorizzato nel database di Prelude Manager.

Se la sorgente dell'attacco risulta essere un server della lan viene mandato un messaggio all'amministratore della sicurezza, quindi si forza il logout dell'utente responsabile dell'attacco. Se tale utente risulta non essere dotato di diritto di superutente gli viene revocato qualsiasi diritto di login prima di mandare un rapporto dettagliato all'amministratore della sicurezza. Se l'utente risulta essere dotato di diritti di supreutente viene attivato il quinto livello decisionale di SLUNP (vedi sezione 5.2.29).

5.2.27 Modulo worms 03

L'albero decisionale di figura 5.34 ci mostra i diversi comportamenti che il sistema esperto SLUNP tiene a seconda che la sorgente dell'attacco risulti essere un host oppure un server della rete locale.

Se si tratta di un semplice host viene mandato un messaggio di avvertimento a tutti gli utenti della lan, viene avvertito l'amministratore della sicurezza, viene impedito all'host infettato qualsiasi tipo di connessione di rete ed infine viene creato un allarme che verrà memorizzato all'interno del database di Prelude Manager.

Se la sorgente dell'attacco risulta essere un sever della rete locale viene richiesto l'intervento dell'amministratore della sicurezza poiché il sistema esperto ha raggiunto i propri limiti decisionali. Le sole opzioni che ha a disposizione SLUNP sono quelle di dare ulterriore priorità ai processi che hanno il compito di uccidere i processi worms e di attivare metodi alternativi per negare le connessioni di rete ai processi worms.

5.2.28 Modulo D.D.o.S. 03

La figura 5.35 ci mostra come venga data importanza ad un tipo di reazione attiva nei confronti della sorgente dell'attacco: vengono infatti attivati dei tool in grado di rilevare le vulnerabilità del sistema attaccante, quindi viene richiesta l'interazione con l'amministratore della sicurezza che deve decidere se procedere con l'attivazione delle contromisure suggerite o meno. Qualunque sia la sua decisione viene creato un allarme per lo storico del database centrale.

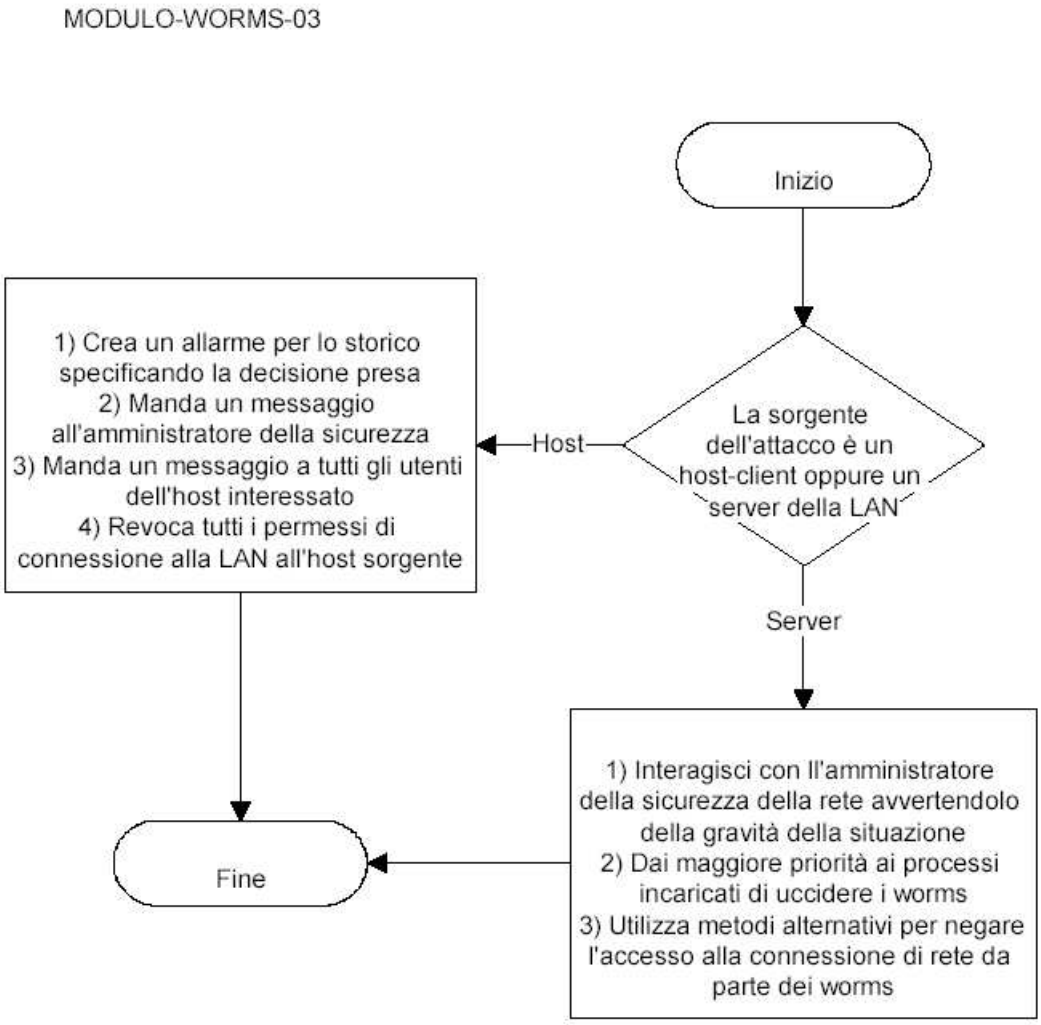

Figura 5.34: Albero decisionale del modulo worms 03

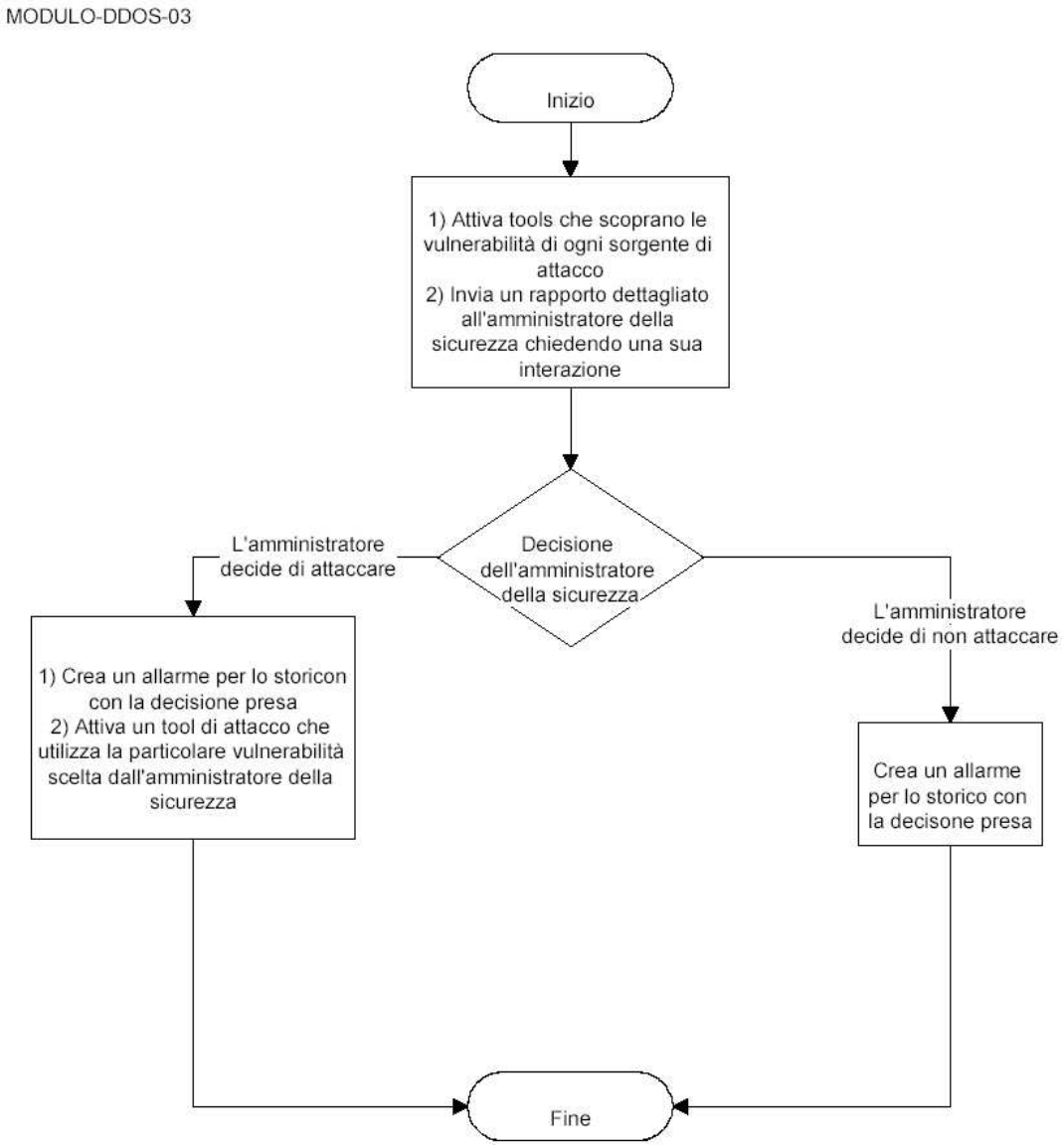

Figura 5.35: Albero decisionale del modulo D.D.o.S. 03

5.2.29 Livello 3: quinto livello decisionale di SLUNP

Questo livello decisionale, il cui albero viene mostrato in figura 5.36, viene utilizzato solamente per la verifica dell'autenticità del superutente loggato nel sistema. Se il tool che ha il compito di verificare tale autenticità dà esito positivo viene creato un semplice allarme per lo storico del databse centrale, altrimenti viene avvertito l'amministratore della sicurezza e vengono revocati i diritti di login dell'utente. Dopo questa prima fase di verifica viene deciso se attiare ulteriori reazioni all'intrusione o meno; tale decisione viene presa in base all'importanza del server e della rete locale. Se è giudicata bassa non vengono intraprese ulteriori azioni, alrimenti viene attivato un backtracing per risalire alla sorgente dell'attacco, viene mandata una mail di avvertimento.

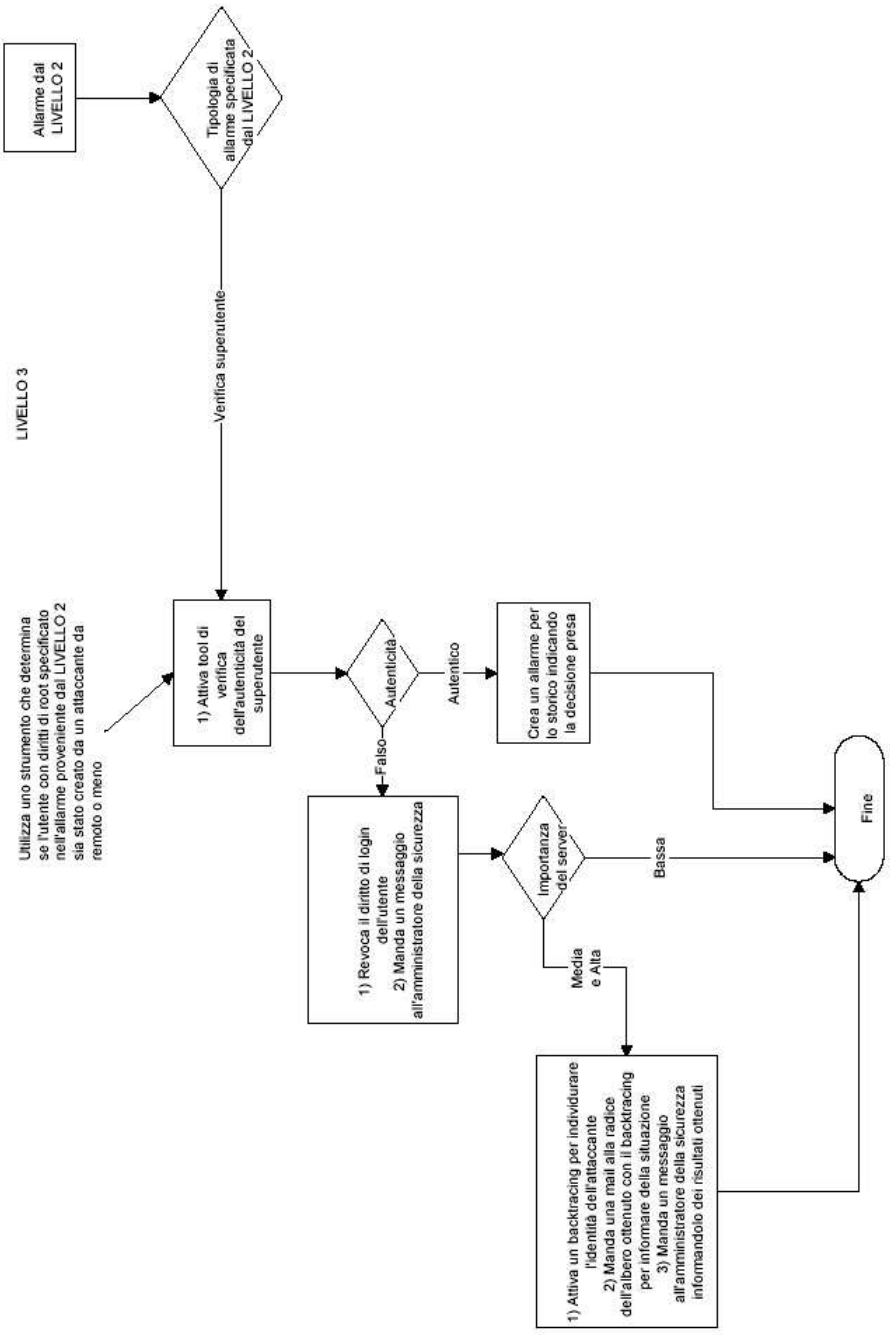

Figura 5.36: Albero decisionale del livello 03

Capitolo 6

Implementazione dell'Inteligent IDS

In questo capitolo viene presentato il lavoro sperimentale svolto in questa tesi. Basandosi su quanto descritto nella sezione 5.2 e successive sottosezioni vengono ora presentate le parti di maggiore interesse del sistema realizzato.

Dopo un'attenta analisi delle varie tipologie di attacco elencate nelle tabelle 5.1 e 5.2, si è deciso di implementare gli alberi decisionali relativi alla difesa dagli attacchi di tipo *port scanning* ed *ipflood*, nonché gli alberi decisionali creati per gestire i *sovraccarichi di sistema*. In ciascuna delle sottosezioni seguenti viene presentato un blocco di regole di CLIPS (vedi sezione 2.1 e capitolo 3) relativo ad un livello decisionale o ad un attacco specifico; nella descrizione di quanto realizzato verranno effettuati riferimenti agli alberi decisionali a cui il codice presentato si riferisce in modo da facilitare la comprensione del lavoro svolto.

6.1 Istruzione del sistema esperto S.L.U.N.P.

Durante la lettura di queste sezioni bisogna sempre tenere presente che SLUNP non è un sistema eperto a sé creato ad hoc per essere integrato con Prelude IDS, non è cioè possibile paragonarlo ad un programma scritto in C che viene compilato ed eseguito autonomamente. Si tratta bensì di una serie di regole scritte nello standard previsto da CLIPS, regole che vengono caricate all'interno di CLIPS e quindi eseguite al fine di ottenere una decisione seguendo un determinato percorso.

Avendo ben presente questo punto fondamentale possiamo ora passare alla descrizione del sistema esperto SLUNP.

Prima di poter utilizzare gli alberi decisionali costruiti nella progettazione teorica,

è necessario fornire al sistema esperto tutti i dati e le classi che gli serviranno per funzionare.

Come prima cosa viene utilizzata la programmazione Object Oriented presente all'interno di CLIPS al fine di creare una nuova classe i cui campi rispecchiano fedelmente le varie classi presenti in un messaggio IDMEF, con le relative molteplicità ed i rispettivi valori di default (qualora siano previsti). Vengono inoltre fornite a SLUNP le conoscenze necessarie per riuscire ad identificare lo scenario in cui deve operare.

```
(defclass IDMEF (is-a USER) (role concrete) (pattern-match reactive)
(slot idmef-type (create-accessor read-write) (pattern-match reactive) (default "alert"))
(slot analyzer-id (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot analyzer-manufacturer (create-accessor read-write) (pattern-match reactive))
(slot analyzer-model (create-accessor read-write) (pattern-match reactive))
(slot analyzer-version (create-accessor read-write) (pattern-match reactive))
(slot analyzer-class (create-accessor read-write) (pattern-match reactive))
(slot analyzer-ostype (create-accessor read-write) (pattern-match reactive))
(slot analyzer-osversion (create-accessor read-write) (pattern-match reactive))
(slot analyzer-node-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot analyzer-node-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot analyzer-node-location (create-accessor read-write) (pattern-match reactive))
(slot analyzer-node-name (create-accessor read-write) (pattern-match reactive))
(slot analyzer-node-address-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot analyzer-node-address-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot analyzer-node-address-vlan-name (create-accessor read-write) (pattern-match reactive))
(slot analyzer-node-address-vlan-num (create-accessor read-write) (pattern-match reactive))
(slot analyzer-node-address-name (create-accessor read-write) (pattern-match reactive))
(slot analyzer-node-address-address (create-accessor read-write) (pattern-match reactive))
(slot analyzer-process-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot analyzer-process-name (create-accessor read-write) (pattern-match reactive))
(slot analyzer-process-pid (create-accessor read-write) (pattern-match reactive))
(slot analyzer-process-path (create-accessor read-write) (pattern-match reactive))
(multislot analyzer-process-arg (create-accessor read-write) (pattern-match reactive))
(multislot analyzer-process-env (create-accessor read-write) (pattern-match reactive))
(multislot analyzertime (create-accessor read-write) (pattern-match reactive))
(multislot createtime (create-accessor read-write) (pattern-match reactive))
(multislot detecttime (create-accessor read-write) (pattern-match reactive))
(slot additionaldata-type (create-accessor read-write) (pattern-match reactive) (default "string"))
(multislot additionaldata-meaning (create-accessor read-write) (pattern-match reactive))
(slot assessment-impact-severity (create-accessor read-write) (pattern-match reactive))
(slot assessment-impact-completion (create-accessor read-write) (pattern-match reactive))
(slot assessment-impact-type (create-accessor read-write) (pattern-match reactive) (default "other"))
(slot assessment-impact-description (create-accessor read-write) (pattern-match reactive))
(slot assessment-action-category (create-accessor read-write) (pattern-match reactive) (default "other"))
(slot assessment-confidence-rating (create-accessor read-write) (pattern-match reactive) (default "0.9"))
(slot classification-origin (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot classification-name (create-accessor read-write) (pattern-match reactive))
(slot classification-url (create-accessor read-write) (pattern-match reactive))
(slot source-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot source-decoy (create-accessor read-write) (pattern-match reactive))
(slot source-spoofed (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot source-interface (create-accessor read-write) (pattern-match reactive))
(slot source-node-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot source-node-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
```

```
(slot source-node-location (create-accessor read-write) (pattern-match reactive))
(slot source-node-name (create-accessor read-write) (pattern-match reactive))
(slot source-node-address-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot source-node-address-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot source-node-address-vlan-name (create-accessor read-write) (pattern-match reactive))
(slot source-node-address-vlan-num (create-accessor read-write) (pattern-match reactive))
(slot source-node-address-name (create-accessor read-write) (pattern-match reactive))
(slot source-node-address-address (create-accessor read-write) (pattern-match reactive))
(slot source-user-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot source-user-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(multislot source-user-userid-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(multislot source-user-userid-type (create-accessor read-write) (pattern-match reactive) (default "original-user"))
(slot source-user-userid-name (create-accessor read-write) (pattern-match reactive))
(slot source-userid-number (create-accessor read-write) (pattern-match reactive))
(slot source-process-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot source-process-name (create-accessor read-write) (pattern-match reactive))
(slot source-process-pid (create-accessor read-write) (pattern-match reactive))
(slot source-process-path (create-accessor read-write) (pattern-match reactive))
(multislot source-process-arg (create-accessor read-write) (pattern-match reactive))
(multislot source-process-env (create-accessor read-write) (pattern-match reactive))
(slot source-service-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot source-service-name (create-accessor read-write) (pattern-match reactive))
(slot source-service-port (create-accessor read-write) (pattern-match reactive))
(slot source-service-portlist (create-accessor read-write) (pattern-match reactive))
(slot source-service-protocol (create-accessor read-write) (pattern-match reactive))
(slot source-service-webservice-url (create-accessor read-write) (pattern-match reactive))
(slot source-service-webservice-cgi (create-accessor read-write) (pattern-match reactive))
(slot source-service-webservice-httpmethod (create-accessor read-write) (pattern-match reactive))
(slot source-service-webservice-arg (create-accessor read-write) (pattern-match reactive))
(slot source-service-snmpservice-oid (create-accessor read-write) (pattern-match reactive))
(slot source-service-snmpservice-community (create-accessor read-write) (pattern-match reactive))
(slot source-service-snmpservice-command (create-accessor read-write) (pattern-match reactive))
(slot target-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot target-spoofed (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot target-interface (create-accessor read-write) (pattern-match reactive))
(slot target-node-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot target-decoy (create-accessor read-write) (pattern-match reactive))
(slot target-node-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot target-node-location (create-accessor read-write) (pattern-match reactive))
(slot target-node-name (create-accessor read-write) (pattern-match reactive))
(slot target-node-address-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot target-node-address-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(slot target-node-address-vlan-name (create-accessor read-write) (pattern-match reactive))
(slot target-node-address-vlan-num (create-accessor read-write) (pattern-match reactive))
(slot target-node-address-name (create-accessor read-write) (pattern-match reactive))
(slot target-node-address-address (create-accessor read-write) (pattern-match reactive))
(slot target-user-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot target-user-category (create-accessor read-write) (pattern-match reactive) (default "unknow"))
(multislot target-user-userid-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(multislot target-user-userid-type (create-accessor read-write) (pattern-match reactive) (default "original-user"))
(slot target-user-userid-name (create-accessor read-write) (pattern-match reactive))
(slot target-userid-number (create-accessor read-write) (pattern-match reactive))
(slot target-process-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(slot target-process-name (create-accessor read-write) (pattern-match reactive))
(slot target-process-pid (create-accessor read-write) (pattern-match reactive))
(slot target-process-path (create-accessor read-write) (pattern-match reactive))
(multislot target-process-arg (create-accessor read-write) (pattern-match reactive))
(multislot target-process-env (create-accessor read-write) (pattern-match reactive))
(slot target-service-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
```

```
(slot target-service-name (create-accessor read-write) (pattern-match reactive))
(slot target-service-port (create-accessor read-write) (pattern-match reactive))
(slot target-service-portlist (create-accessor read-write) (pattern-match reactive))
(slot target-service-protocol (create-accessor read-write) (pattern-match reactive))
(slot target-service-webservice-url (create-accessor read-write) (pattern-match reactive))
(slot target-service-webservice-cgi (create-accessor read-write) (pattern-match reactive))
(slot target-service-webservice-httpmethod (create-accessor read-write) (pattern-match reactive))
(slot target-service-webservice-arg (create-accessor read-write) (pattern-match reactive))
(slot target-service-snmpservice-oid (create-accessor read-write) (pattern-match reactive))
(slot target-service-snmpservice-community (create-accessor read-write) (pattern-match reactive))
(slot target-service-snmpservice-command (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-name (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-path (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-createtime (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-modifytime (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-accesstime (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-disksize (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-datasize (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-ident (create-accessor read-write) (pattern-match reactive) (default "0"))
(multislot target-filelist-file-category (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-fstype (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-fileaccess-userid (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-fileaccess-permission (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-linkage-category (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-linkage-name (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-linkage-path (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-linkage-file (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-inode-changetime (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-inode-number (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-inode-majordevice (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-inode-minordevice (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-inode-cmajordevice (create-accessor read-write) (pattern-match reactive))
(multislot target-filelist-file-inode-cminordevice (create-accessor read-write) (pattern-match reactive)))
```

Il sistema ha ora bisogno di conoscere lo scenario in cui deve operare: a tal fine necessita di conoscere le finestre temporali in cui viene prevista la possibilità di verificarsi di anomalie e l'importanza della rete locale all'interno della quale opera.

```
(assert (evento-previsto "cpu" "2003" "04" "09" "08" "00" "00" "000" "+0100"
"2003" "04" "09" "10" "30" "00" "000" "+0100"))
```

Questo è un esempio di definizione di una finestra temporale: viene fornita a SLUNP la conoscenza del fatto che viene previsto un sovraccarico di CPU tra le 08:00:00,000 +01:00GTM del 09/04/2003 e le 10:30:00.000 +01:00GTM dello stesso giorno.

```
(assert (importanza-server alta))
```

Con questa istruzione il sistema esperto viene messo a conoscenza del fatto che la rete locale all'interno della quale opera è giudicata di importanza molto elevata.

6.2 Livello 0: richiesta di difesa

Il codice che verrà presentato qui di seguito è relativo all'albero decisionale descritto nella sezione 5.2.1. Di tale codice verranno commentate solamente le parti di maggiore interesse e quelle di maggiore difficoltà di comprensione.

Queste regole hanno il compito di inizializzare SLUNP permettendogli di dare inizio ai suoi processi decisionali ed impedendogli di terminarli senza terminare anche l'esecuzione di CLIPS.

```
(defrule inizializza
(declare (salience 99))
?istanza<-(object (is-a IDMEF))
=>(assert (confidenza-ok ?istanza)))
(defrule inizializza-uscita-ok
(object (is-a IDMEF)
(classification-name ?f&~"sovraccarico"&~"SCAN Proxy attempt"))
=>
(assert (uscita-automatica on)))
(defrule uscita-ok
(declare (salience -99))
(uscita-automatica on)
=>
(exit))
```

Si chiedere conferma della contromisura che il sensore ha eventualmente attivato in modo automatico, andando ad agire direttamente sul firewall. Se il sensore non ha attivato alcuna reazione automatica (o se non ha la possibilità di iteragire direttamente con il firewall) all'utente non verrà richiesta la conferma.

```
(defrule conferma-azione-sensore-locale
(declare (salience 97))
?nsovraccarico<-(object (is-a IDMEF)
(additionaldata-meaning ?azione&~"no azione del sensore" $?)
(analyzer-process-name "prelude-lml"))
(confidenza-ok ?nsovraccarico)
=>(printout t "Rilevato allarme di sovraccarico di tipo locale" crlf)
(printout t (send (instance-name ?nsovraccarico) get-additionaldata-meaning) crlf)
(printout t "Conferma dell'azione intrapresa? (s/n)" crlf)
(bind ?conferma (read))
(assert (risposta-accetta-azione-sensore ?nsovraccarico (lowcase ?conferma))))
(defrule verifica-risposta-1
(declare (salience 99))
?fatto<-(risposta-accetta-azione-sensore ? ?risposta&~s&~n)
=>
(printout t "Inserita risposta errata" crlf)
(retract ?fatto)
(refresh conferma-azione-sensore-locale))
(defrule verifica-risposta-3
(declare (salience 99))
?fatto<-(risposta-accetta-azione-sensore ?istanza ?risposta&s|n)
=>
(retract ?fatto)
```

```
(assert (accetta-azione-sensore ?istanza ?risposta)))
```

All'utente viene richiesto se, oltre all'eventuale azione del sensore (che può essere accettata o meno) ha intenzione di richiedere a SLUNP di identificare ulteriori contromisure da adottare in relazione all'attacco rilevato.

```
(defrule verifica-risposta-4
(declare (salience 99))
?fatto<-(risposta-azioni-supplementari ?istanza ?risposta&s|n)
=>
(retract ?fatto)
(assert (azioni-supplementari ?istanza ?risposta)))
(defrule verifica-risposta-2
(declare (salience 99))
?fatto<-(risposta-azioni-supplementari ? ?risposta&~s&~n)
=>
(printout t "Inserita risposta errata" crlf)
(retract ?fatto)
(refresh richiedi-azioni-supplementari))
(defrule richiedi-azioni-supplementari
(declare (salience 95))
(confidenza-ok ?istanza)
;(accetta-azione-sensore ?istanza ?)
=>
(printout t "Richiesta di ulteriori azioni per l'istanza " ?istanza " (s/n) ?" crlf)
(bind ?conferma (read))
(assert (risposta-azioni-supplementari ?istanza (lowcase ?conferma))))
```

Se non si intende procedere oltre, SLUNP termina il proprio processo decisionale; viene terminata anche l'esecuzione di CLIPS.

```
(defrule no-azioni-supplementari
(declare (salience 94))
(azioni-supplementari ?istanza n)
=>
(exit))
```

Questa regola viene eseguita se si decide di annullare l'eventuale contromisura che il sensore ha attivato in automatico modificando le regoel del firewall.

```
(defrule annulla-azione-sensore
(declare (salience 96))
?f<-(accetta-azione-sensore ?istanza ?conferma&n)
=>
(bind ?buf (send (instance-name ?istanza) get-additionaldata-meaning))
(printout t "Attivare attuatore che annulla la seguente azione del sensore: " crlf)
(printout t ?buf crlf))
```

6.3 Modulo 1: scelta delle difese

Il codice seguente è relativo all'albero decisinale presentato nella sezione 5.2.2.

Come prima operazione, SLUNP traduce il valore del livello di confidenza eventualmente specificato dal sensore in un dato di tipo integer.

```
(defrule traduci-livello-confidenza-low
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating "low"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.1))
(defrule traduci-livello-confidenza-medium
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating "medium"))=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.5))
(defrule traduci-livello-confidenza-high
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating "high"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.9))
(defrule livello-confidenza-numerico-00
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.0"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.0))
(defrule livello-confidenza-numerico-01
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.1"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.1))
(defrule livello-confidenza-numerico-02
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.2"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.2))
(defrule livello-confidenza-numerico-03
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.3"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.3))
(defrule livello-confidenza-numerico-04
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.4"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.4))
(defrule livello-confidenza-numerico-05
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.5"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.5))
(defrule livello-confidenza-numerico-06
(declare (salience 98))
```

```
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.6"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.6))
(defrule livello-confidenza-numerico-07
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.7"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.7))
(defrule livello-confidenza-numerico-08
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.8"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.8))
(defrule livello-confidenza-numerico-09
(declare (salience 98))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&"0.9"))
=>
(send (instance-name ?istanza) put-assessment-confidence-rating 0.9))
```

Il codice presentato di seguito viene utilizzato per implementare la parte dell'albero decisionale di figura 5.6 che prevede la selezione della tipologia di attacco *sovraccarico di sistema* ed il controllo che l'anomalia registrata rientri o meno in una finestra temporale.

```
(defrule verifica-finestra-temporale
?istanza<-(object (is-a IDMEF)
(classification-name "sovraccarico")
(classification-origin ?origine)
(detecttime ?aa ?mm ?gg ?hh ?min ?sec ?milsec ?gtm))
(confidenza-ok ?istanza)
(evento-previsto ?origine ?aai ?mmi ?ggi ?hhi ?mini ?seci ?milseci ?gtmi
?aaf ?mmf ?ggf ?hhf ?minf ?secf ?milsecf ?gtmf)
=>
(bind ?aaci (str-compare ?aai ?aa))
(bind ?mmci (str-compare ?mmi ?mm))
(bind ?ggci (str-compare ?ggi ?gg))
(bind ?hhci (str-compare ?hhi ?hh))
(bind ?minci (str-compare ?mini ?min))
(bind ?secci (str-compare ?seci ?sec))
(bind ?milsecci (str-compare ?milseci ?milsec))
(bind ?gtmci (str-compare ?gtmi ?gtm))
(bind ?aacf (str-compare ?aa ?aaf))
(bind ?mmcf (str-compare ?mm ?mmf))
(bind ?ggcf (str-compare ?gg ?ggf))
(bind ?hhcf (str-compare ?hh ?hhf))
(bind ?mincf (str-compare ?min ?minf))
(bind ?seccf (str-compare ?sec ?secf))
(bind ?milseccf (str-compare ?milsec ?milsecf))
(bind ?gtmcf (str-compare ?gtm ?gtmf))
(assert (finestra-temporale ?istanza ?aaci ?mmci ?ggci ?hhci ?minci
?secci ?milsecci ?gtmci ?aacf ?mmcf ?ggcf ?hhcf ?mincf ?seccf ?milseccf ?gtmcf)))
(defrule normaalizza-controllo-finestra-temporale-1
```

```
(declare(salience 1))
?fatto<-(finestra-temporale ?istanza ?aaci&0 ?mmci&0 ?ggci&0 ?hhci&-1 ?minci
?secci ?milsecci ?gtmci&0 ?aacf&0 ?mmcf&0 ?ggcf&0 ?hhcf&-1 ?mincf ?seccf ?milseccf ?gtmcf&0)
=>
(retract ?fatto)
(assert (finestra-temporale ?istanza 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(defrule normaalizza-controllo-finestra-temporale-2
(declare(salience 1))
?fatto<-(finestra-temporale ?istanza ?aaci&0 ?mmci&0 ?ggci&0 ?hhci&0 ?minci&~1
?secci ?milsecci ?gtmci&0 ?aacf&0 ?mmcf&0 ?ggcf&0 ?hhcf&-1 ?mincf ?seccf ?milseccf ?gtmcf&0)
=>
(retract ?fatto)
(assert (finestra-temporale ?istanza 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(defrule normaalizza-controllo-finestra-temporale-3
(declare(salience 1))
?fatto<-(finestra-temporale ?istanza ?aaci&0 ?mmci&0 ?ggci&0 ?hhci&-1 ?minci
?secci ?milsecci ?gtmci&0 ?aacf&0 ?mmcf&0 ?ggcf&0 ?hhcf&0 ?mincf&~1 ?seccf ?milseccf ?gtmcf&0)
=>
(retract ?fatto)
(assert (finestra-temporale ?istanza 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(defrule normaalizza-controllo-finestra-temporale-4
(declare(salience 1))
?fatto<-(finestra-temporale ?istanza ?aaci&0 ?mmci&0 ?ggci&0 ?hhci&0 ?minci&~1
?secci ?milsecci ?gtmci&0 ?aacf&0 ?mmcf&0 ?ggcf&0 ?hhcf&0 ?mincf&~1 ?seccf ?milseccf ?gtmcf&0)
=>
(retract ?fatto)
(assert (finestra-temporale ?istanza 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
```

Se l'allarme è relativo ad un evento che rientra in una finestra temporale prevista SLUNP termina i suoi processi decisionali attivando la procedura di uscita da CLIPS.

```
(defrule elimina-allarmi-entro-finestra-temporale
(declare (salience 85))
?fatto<-(finestra-temporale ?istanza ?a&0 ?b&0 ?c&0 ?d&~1 ?e&~1 ?f&~1 ?g&~1 ?h&0 ?i&0 ?l&0
?m&0 ?n&~1 ?o&~1 ?p&~1 ?q&~1 ?r&0)
=>
(printout t "L'allarme " (instance-name ?istanza) "rientra in una finestra temporale" crlf)
(exit))
```

Se siamo di fronte ad un evento non previsto si attiva il backtracing della sorgente dell'attacco e si attiva il terzo livello decisionale di SLUNP (vedi sezione 5.2.4).

```
(defrule attiva-tracing-sistema
(declare (salience -2))
?istanza<-(object (is-a IDMEF)
(classification-name "sovraccarico"))
(confidenza-ok ?istanza)
=>(printout t "attivare-tracing per "?istanza)
(assert (attivare-livello1 ?istanza))
)
```

6.4 Livello 0b: incremento della fiducia

Il codice presentato in questa sezione implementa l'albero decisionale di figura 5.7.

```
(defrule verifica-livello-confidenza
(declare (salience 99))
?istanza<-(object (is-a IDMEF)
(assessment-confidence-rating ?conf&0.0|0.1|0.2|0.3|0.4))
?fatto<-(confidenza-ok ?istanza)
=>
(assert (verifica-confidenza-allarme))
(retract ?fatto)
(printout t "Attivo LIVELLOOB per l'istanza " (instance-name ?istanza) crlf))
```

Questa regola viene utilizzata per attivare i tool che hanno il compito di effettuare nuove analisi dei dati forniti dal sensore al fine di incrementare il livello di confidenza circa l'allarme rilevato (oppure specificare il nuovo tipo di attacco che sta minacciando realmente la rete locale). Poiché tali strumenti di analisi non sono stati ancora realizzati, né erano oggetto di studio di questa tesi, durante gli esperimenti svolti utilizzando SLUNP si è solamente potuto simulare l'effetto che tali tool di analisi hanno sul processo decisionale di SLUNP.

```
(defrule incremento-livello-confidenza-sovraccarico-cpu
(declare (salience 98))
?fatto<-(verifica-confidenza-allarme)
?istanza<-(object (is-a IDMEF)
(classification-origin "cpu")
(classification-name "sovraccarico"))
=>
(printout t "attiva tool di incremento del livello di confidenza per un sovraccarico di CPU" crlf)
(retract ?fatto)
(send (instance-name ?istanza) put-assessment-confidence-rating "high")
(assert (confidenza-ok ?istanza)))
(defrule incremento-livello-confidenza-port-scan
(declare (salience 98))
?fatto<-(verifica-confidenza-allarme)
?istanza<-(object (is-a IDMEF)
(classification-name "SCAN Proxy attempt"))
=>
(printout t "attiva tool di incremento del livello di confidenza per un port scan" crlf)
(retract ?fatto)
(send (instance-name ?istanza) put-assessment-confidence-rating "high")
(assert (confidenza-ok ?istanza))
(refresh primo-port-scan)
(refresh primo-port-scan-seconda-parte)
(refresh primo-port-scan-esci)
(refresh primo-port-scan-seconda-parte-esegui)
(refresh attendi-livello-2)
(refresh invoco-la-benedizione-di-budda)
(refresh gto)
(refresh lain)
(refresh verifica-superutente)
(refresh kill-primo-tempo)
(refresh kill-secondo-tempo))
```

6.5 Modulo 2: rilevazione e gestione sovraccarico di sistema, rilevazione e gestione ip-flooding

Il seguente codice implementa l'albero decisionale di figura 5.9.

Poiché non esistono tool di verifica dello stato della rete durante gli esperimento svolti si è dovuto simulare vari stati della rete interna e della rete esterna.

```
(defrule attivazione-livello1
(declare (salience -13))
(attivare-livello1 ?istanza)
=>(printout t "Attiva tool per la verifica dello stato della rete rivolta verso Internet" crlf)
(printout t "Attiva tool per la verifica dello stato della rete LAN" crlf)
(printout t "Attiva tool per individuare i processi che sovraccaricano maggiormente il sistema" crlf)
(printout t "Attiva tool per individuare i servizi che sovraccaricano maggiormente il sistema" crlf)
(printout t "Ridistribuisci le risorse di sistema per impedire lo stallo dei processi a bassa priorità" crlf)
(assert (rete-lan-sovraccarica ?istanza)))
(defrule verifica-sovraccarico-rete-lan
(declare (salience -14))
(rete-lan-sovraccarica ?istanza)
=>
(printout t "Rilevato sovraccarico della rete LAN, con rifermento all'allarme: " ?istanza crlf))
(defrule verifica-sovraccarico-rete-esterna
(declare (salience -14))
(rete-esterna-sovraccarica ?istanza)
=>
(printout t "Rilevato sovraccarico proveniente dalla rete esterna, con rifermento all'allarme: " ?istanza crlf))
(defrule rilevamento-attacchi
(declare (salience -15))
(attivare-livello1 ?istanza)
=>(printout t "Attivare tools di analisi dei processi per verificare che non facciano parte")
(printout t " di un qualche tipo di attacco" crlf))
(defrule crea-allarme-IDMEF
(declare (salience -16))
?fatto<-(rilevato-attacco ?istanza ?attacco)
?fatto1<-(confidenza-ok ?istanza)
=>
(retract ?fatto)
(retract ?fatto1)
(send (instance-name ?istanza) put-assessment-confidence-rating "0.9")
(send (instance-name ?istanza) put-additionaldata-meaning ?attacco)
(exit))
(defrule verifica-cpu-hd
(declare (salience -17))
(attivare-livello1 ?istanza)
=>
(printout t "Attivo tool di verifica dell'attuale situazione di CPU ed Hard Disk")
(assert (sovraccarico-cpu-hd "hard disk" ?istanza)))
(defrule terminator
(sovraccarico-cpu-hd ?risorsa ?istanza)
=>
(printout t "Attivare tools di sterminio delle applicazioni che sovraccaricano la risorsa: " ?risorsa crlf)
(printout t "In riferimento all'allarme " ?istanza crlf)
(exit))
```

6.6 Modulo 3: rilevazione e gestione port-scanning

Il codice seguente si riferisce all'implementazione dell'albero decisionale presentato in figura 5.8.

```
(defrule primo-port-scan
(importanza-server ?o&~bassa)
?istanza<-(object (is-a IDMEF)
(classification-name "SCAN Proxy attempt"))
=>
(printout t "Attiva su tutti gli host della rete tools di simulazione di vulnerabilità" crlf)
(printout t "PortScanning in corso" crlf)
(assert (allarme-liv-2 ?istanza)))
(defrule primo-port-scan-seconda-parte
(importanza-server ?o&~bassa)
?u<-(object (is-a IDMEF)
(classification-name "SCAN Proxy attempt")
(source-spoofed "no")
(source-node-address-address ?i)
(allarme-liv-2 ?u)
=>
(assert (sorgente-port-scan ?i (stringp ?i))))
```

Condizione fondamentale per poter fronteggiare al meglio un attacco di questo tipo è quella di avere a disposizione l'indirizzo IP della sorgente dell'attacco; se il sensore non è in grado di fornircelo SLUNP termina i suoi processi decisionali poiché non è in grado di proseguire oltre.

```
(defrule primo-port-scan-esci
(sorgente-port-scan ?i FALSE)
=>
(printout t "Non posso continuare poiché non ho dati sulla sorgente dell'attacco" crlf)
(exit))
(defrule primo-port-scan-seconda-parte-esegui
(sorgente-port-scan ?i TRUE)
=>
(printout t "Attiva tool per monitorare il flusso di dati della sorgente " ?i crlf))
```

6.7 Livello 2: Verifica delle intenzioni dell'attaccante

Il codice presentato di segiuto è relativo all'albero decisionale presentato nella sezione 5.2.25.

Dopo aver atteso per X minuti (il tempo specificato dall'amministratore della sicurezza della lan), si simula il risultato della verifica della presenza di ulteriore traffico sospetto.

```
(defrule attendi-livello-2
(declare (salience -30))
(importanza-server ?o&~bassa)
?istanza<-(object (is-a IDMEF)
(classification-name "SCAN Proxy attempt"))
?fatto<-(allarme-liv-2 ?istanza)
=>
(retract ?fatto)
(printout t "Attesa per X minuti terminata, passo alla reazione" crlf)
(assert (traffico-sospetto ?istanza))
(assert (traffico uscente)))
```

Questa regola permette invece di identificare un problema relativo a quanto rilevato dal sensore: se l'indirizzo della sorgente risulta spoofed ciò significa che l'attaccante non potrà mai conoscere i risultati del port scanning che ha attivato e questo è inconcepibile. E' evidente quindi che il sensore ha commesso un errore di analisi all'atto della crazione dell'allarme e quindi SLUNP deve attivare ulteriori analisi sui dati in suo possesso. Per fare ciò pone nullo il livello di confidenza dell'allarme in modo da forzare l'attivazione immediata delle procedure di analisi dei dati (vedi sezione 6.4).

```
(defrule invoco-la-benedizione-di-budda
(importanza-server ?o&~bassa)
?istanza<-(object (is-a IDMEF)
(source-spoofed ?i&~"no"))
(traffico-sospetto ?istanza)
=>
;(printout t "PortScanning in corso da indirizzo sorgente spoofed" crlf)
(send (instance-name ?istanza) put-assessment-confidence-rating "0.0"))
(defrule gto
(declare (salience -31))
(importanza-server ?o&~bassa)
?istanza<-(object (is-a IDMEF)
(source-spoofed ?i&"no"))
(traffico-sospetto ?istanza)
(traffico uscente)
=>
(printout t "Forza il logout dell'utente " (send (instance-name ?istanza) get-source-user-userid-name) crlf)
(assert (verifica-superutente ?istanza)))
(defrule lain
(declare (salience -31))
(importanza-server ?o&~bassa)
?istanza<-(object (is-a IDMEF)
(source-spoofed ?i&"no"))
(traffico-sospetto ?istanza)
(traffico entrante)
=>
(printout t "Attiva tool di verifica e rapporto delle vulnerabilità dell'attaccante" crlf)
(printout t "Attiva tool per mandare una mail all'amministratore della rete attaccante" crlf)
(printout t "Il firewall rifiuta le connessioni con l'attaccante per Y ore" crlf)
(exit))
```

6.8 Livello 03: verifica superutente

Il codice presentato in questa sezione implementa l'albero decisionale di figura 5.36.

```
(defrule verifica-superutente
?fatto<-(verifica-superutente ?istanza)
=>
(printout t "Attiva tool di verifica dell'utente "(send(instance-name ?istanza)get-source-user-userid-name)crlf)
(assert (utente falso ?istanza)))
(defrule kill-primo-tempo
?fatto<-(utente falso ?istanza)
=>
(retract ?fatto)
(printout t "Revoca i diritti di login all'utente " (send(instance-name ?istanza)get-source-user-userid-name)crlf)
(printout t "Revocati i diritti all'utente " (send (instance-name ?istanza) get-source-user-userid-name) crlf)
(assert (revocato utente ?istanza)))
(defrule kill-secondo-tempo
(importanza-server ?o&~bassa)
?fatto<-(revocato utente ?istanza)
=>
(retract ?fatto)
(printout t "attiva backtracing dell'attaccante" crlf)
(exit))
(defrule kill-terzo-tempo
(importanza-server bassa)
(revocato utente ?istanza)
=>
(exit))
```

6.9 Integrazione fra Prelude IDS e SLUNP

Durante la fase di realizzazione è sorto il problema di trovare un metodo rapido e computazionalmente poco oneroso per integrare la teconologia del sistema esperto SLUNP all'interno di Prelude IDS. Si è scelto di sfruttare il più possibile la struttura già ottimizzata per le applicazioni di tipo distribuito di Prelude IDS che, oltre a risultare veloce e sicura, sfrutta le potenzialità di Libprelude (vedi sezione 4.1.2.1) per la gestione delle comunicazioni fra sensori e manager. E' stato quindi analizzato nel dettaglio il codice sorgente di Prelude Manager (*prelude-manager.c*) e del plug-in responsabile del report testuale di tutti gli allarmi ricevuti dal manager (*textmod.c*). Sono state identificate le istruzioni relative all'apertura del file testuale specificato all'interno del file di configurazione di Prelude Manager e sono state modificate al fine di far sì che all'interno del file testuale sia presente solamente l'ultimo allarme ricevuto dal manager.

Le modifiche apportate al plug-in sopra citato riguardano la chiamata ad un programma esterno che ha il compito di effettuare un primo parsing del file testuale

di log traducendolo in un formato creato ad hoc durante la progettazione di SLUNP. Il file testuale così prodotto subisce un'ulteriore fase di parsing al fine di tradurre le informazioni contenute nell'allarme del sensore in istruzioni utilizzabili da SLUNP per creare un'istanza della classe IDMEF (vedi sezione 6.1).

Terminate queste due fasi di parsing viene creato un unico file di testo contenente sia il codice introdotto nelle precedenti sezioni di questo capitolo, sia l'istanza appena creata; viene quindi attivato il sistema esperto CLIPS indicando il fatto che le regole che dovrà utilizzare non verranno inserite utilizzando la shell a riga di comando, ma dovranno essere ricercate all'interno del file di testo appena creato.

Il controllo viene quindi lasciato a CLIPS fino a quando SLUNP non termina i suoi processi decisionali.

Di seguito viene presentato parte del codice scritto per realizzare l'integrazione fra Prelude IDS e SLUNP.

La procedura presentata viene utilizzata da Prelude Manager per attivare le procedure di traduzione dell'allarme ricevuto dal formato IDMEF ad un formato testo.

```
static void process_message(const idmef_message_t *msg)
{
char c = EOF;
char buf[120];
int nread = 0;
```

Prima di procedere alla traduzione, il file che deve contenere gli allarmi in formato testo (*/var/log/prelude.log*) viene svuotato; in questo modo esso conterrà solamente l'ultimo allarme ricevuto dal manager. Sebbene la soluzione più semplice per ottenere la cancellazione di un file sia quella di chiude il file per poi riaprirlo utilizzando la modalità di letura-scrittura distruttiva (*fopen("nome-file" "w+");*), in questo caso questa procedura non è applicabile poiché il file prelude.log deve essere mantenuto aperto fino al termine dell'esecuzione di Prelude-Manager. Per questo motivo si è optato per un procedimento alternativo che prevede l'utilizzo del comando cat della bash di Linux. Tale comando è in grado di visualizzare l'intero contenuto di un file, anche se questo è mantenuto aperto da un'altra applicazione. Per ottenere la cancellazione del file prelude.log si è quindi creato un file vuoto, (*prelude-null.log*), quindi il suo contenuto viene visualizzato con il comando cat il cui output viene ridirezionato all'interno del file prelude.log. In questo modo anche quest'ultimo file risulterà vuoto.

Tale procedura non modifica il valore dell'indicatore di posizione del file, un puntatore il cui valore indica la posizione successiva all'ultimo byte del file. E' quindi necessario riposizionale tale indicatore all'inizio del file; questo viene realizzato utilizzando il comando fseek.

```
system("cat /var/log/prelude-null.log > /var/log/prelude.log");
fseek(out_fd, 0L, 0);
```

Questo è il codice originale della procedura process_message che ha il compito di riconoscere se l'allarme in formato IDMEF è di tipo heartbeat oppure di tipo allert e di attivare le relative procedure di traduzione in formato testuale.

```
switch (msg->type) {
case idmef_alert_message:
process_alert(msg->message.alert);
break;
case idmef_heartbeat_message:
process_heartbeat(msg->message.heartbeat);
break;
default:
log(LOG_ERR, "unknow message type: %d.\n", msg->type);
break;
}
fflush(out_fd);
```

Terminata la conversione in formato testuale, il contenuto del file prelude.log viene copiato all'interno di un altro file testuale, *prelude-mio.log*. Tale operazione è resa necessaria dal fatto che il file prelude.log non può essere chiuso, come spiegato in precedenza. Viene quindi chiamato il programma livello0 che attiva tutte le procedure di parsing del file prelude-mio.log, nonché la costruzione del file di istruzioni di CLIPS e l'attivazione di SLUNP.

```
system("cat /var/log/prelude.log > /var/log/prelude-mio.log");
system("./livello0 istruzioni.clp");
}
```

E' stata compiuta un'altra modifica al codice originale di textmod.c, relativa alla prima apertura del file prelude.log. Tale file, che originariamente veniva aperto in modo *append*, viene ora aperto in modo di *lettura-scrittura distruttiva*, come mostrato dal codice seguente.

```
static int set_logfile(prelude_option_t *opt, const char *arg)
{
out_fd = fopen(arg, "w+");
```

```
if ( ! out_fd ) {
log(LOG_ERR, "error opening %s in write mode.\n", arg);
return prelude_option_error;
}
logfile = strdup(arg);
return prelude_option_success;
}
```

Capitolo 7

Conclusioni

Dopo aver effettuato una serie di simulazioni prevedendo vari scenari all'interno dei quali SLUNP deve operare, si sono raggiunti ottimi risultati: il tempo necessario al sistema per passare dall'allarme contenuto all'interno del file di log gestito dal plug-in di Prelude alla richiesta di conferma che SLUNP fa all'utente (vedi sezione 6.2) si aggira nell'ordine di una decina di secondi, mentre il tempo necessario a SLUNP per giungere alla fine dei suoi processi decisionali, se non si tiene conto del ritardo introdotto dall'attesa della risposta dell'utente alle richieste di conferma del sistema, è nell'ordine di qualche decimo di secondo. Questi risultati permettono di affermare che il sistema sviluppato risponde al meglio alle richieste di velocità e semplicità insite nella progettazione teorica, costituendo di fatto un valido strumento per il miglioramento del livello di sicurezza delle reti.

Nonostante gli ottimi risultati esistono comunque alcuni punti che presentano problemi: innanzitutto bisogna chiarire che l'efficienza di SLUNP è strettamente legata al livello di fiducia che l'utente ha nei suoi confronti. Infatti, se l'utente ritiene che le decisioni prese dal sistema esperto debbano essere verificare prima di diventare definitive, i tempi decisionali crescono in modo incontrollato poiché quasi tutte le decisioni devono avere la conferma dell'utente; ciò obbliga SLUNP ad aspettare che l'utente raggiunga una decisione prima di procedere.

Apparentemente questo problema è di facile soluzione: basterebbe infatti dare a SLUNP piena autonomia decisionale, senza prevedere alcuna intromissione da parte dell'utente, per ottenere le massime prestazioni possibili. In realtà ciò non è realizzabile, non tanto per limiti tecnologici, quanto per limitazioni di carattere giuridico: senza alcuna richiesta di conferma all'amministratore della rete locale in cui opera,

le responsabilità legali relative agli effetti delle contromisure attiate ricadrebbero totalmente sugli sviluppatori del sistema di sicurezza della rete poiché sono loro a decidere, durante la progettazione del sistema, quali linee di condotta tenere a fronte di ciascun tipo di attacco. E' chiaro come questo scenario sia totalemente inaccettabile, quindi è necessario un deteminato livello di interazione fra il sistema e l'utente.

Una soluzione accettabile che costituisce un valido compromesso fra prestazioni e carico di responsabiltà è costituita dalla limitazione di questo livello di interazione: SLUNP richiederà l'approvazione dell'utente solamente nei casi di amggiore gravità o prima di attivare contromisure particolari.

Vi sono infine altre due problematiche che incidono in modo rilevante sulle prestazioni di SLUNP. Entrambe sono causate da limitazioni del sistema esperto CLIPS: questo infatti prevede che il codice caricato da file di testo venga visualizzato a video, come se venisse inserito tramite riga di comando. Come è facile immaginare, costituisce un inutile spreco di tempo essendo di diversi ordini di grandezza più lenta rispetto alla velocità computazionale della CPU. La seconda problematica è la maggiore limitazione al sistema sviluppato: CLIPS infatti non permette di effettuare chiamate a programmi esterni, ciò significa che SLUNP non ha in realtà la possibilità di attivare nessuna delle procedure che identifica attraverso i suoi processi decisionali. Una prima soluzione, anche se molto inefficiente, potrebbe essere costituita dall'introduzione di un file testuale in cui vengono salvate alcune stringe particolari che permetteranno ad un successivo filtro di analisi di attivare le contromisure identificate da SLUNP. Tale soluzione non è stata sperimentata poiché non rientra negli obbiettivi di questa tesi, ma ulteriori sviluppi di SLUNP risolveranno la problematica in oggetto.

Capitolo 8

Riconoscimento di un Port-Scanning

In questo capitolo vengono presentati i risultati di un esperimento condotto utilizzando SLUNP ed NMAP, un tool utilizzato per simulare un attacco di tipo port-scanning. [NMAP] Per effettuare questo esperimento viene utilizzato un unico PC (AMD-K6 450Mhz, con 128Mb di RAM) su cui è installato un sistema operativo Linux (Kernel 2.4.21-0.13mdk). Per simulare un host client su cui è installato Prelude-NIDS ed un host server su cui è installato Prelude-Manager si utilizzando due consolle diverse, una utilizzata per eseguire il manager, l'altra per eseguire il sensore. Per comunicare, entrambi utilizzando l'interfaccia di loop-back del sistema operativo (indirizzo IP 127.0.0.1).

La prima operazione da effettuare è quella di attivare Prelude-Manager, come mostrato in figura 8.1. Viene poi attivato Prelude-NIDS che stabilisce una connessione con Prelude-Manager e che monitora il traffico utilizzando l'interfaccia di rete *eth0* (figure 8.2 e 8.3).

Viene quindi attivato il tool NMAP specificando l'indirizzo IP dell'host su cui effettuare il port-scanning; il risultato dello scan viene mostrato in figura 8.4.

Prelude-NIDS rileva quattro diversi tipi di Port-Scanning, quindi Prelude-Manager attiva per quattro volte SLUNP il quale indicherà ogni volta che il sistema si trova sotto un attacco di tipo Port-Scanning. Inoltre, poiché Prelude-NIDS non fornisce informazioni sul fatto che l'indirizzo IP dell'attaccante sia spoofed o meno, SLUNP effettuerà ogni volta una chiamata ad un tool in grado di incrementare il confidence rating dell'allarme ricevuto dal manager. Tutto questo è mostrato in figura 8.5.

Terminato l'esperimento viene disattivato il sensore il quale visualizza alcune statistiche relative al numero di pacchetti analizzati, come mostrato in figura 8.6.

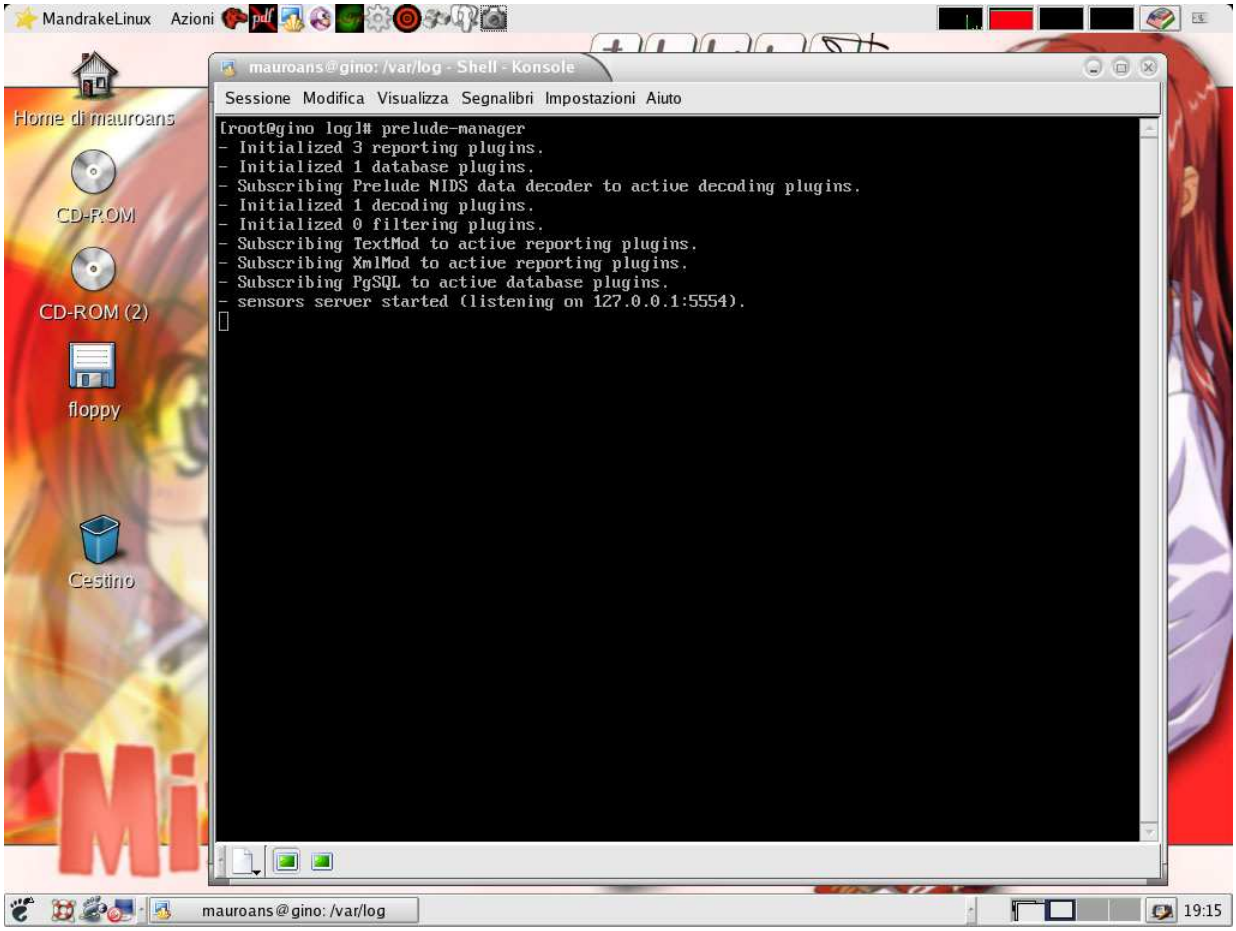

Figura 8.1: Attivazione di Prelude-Manager

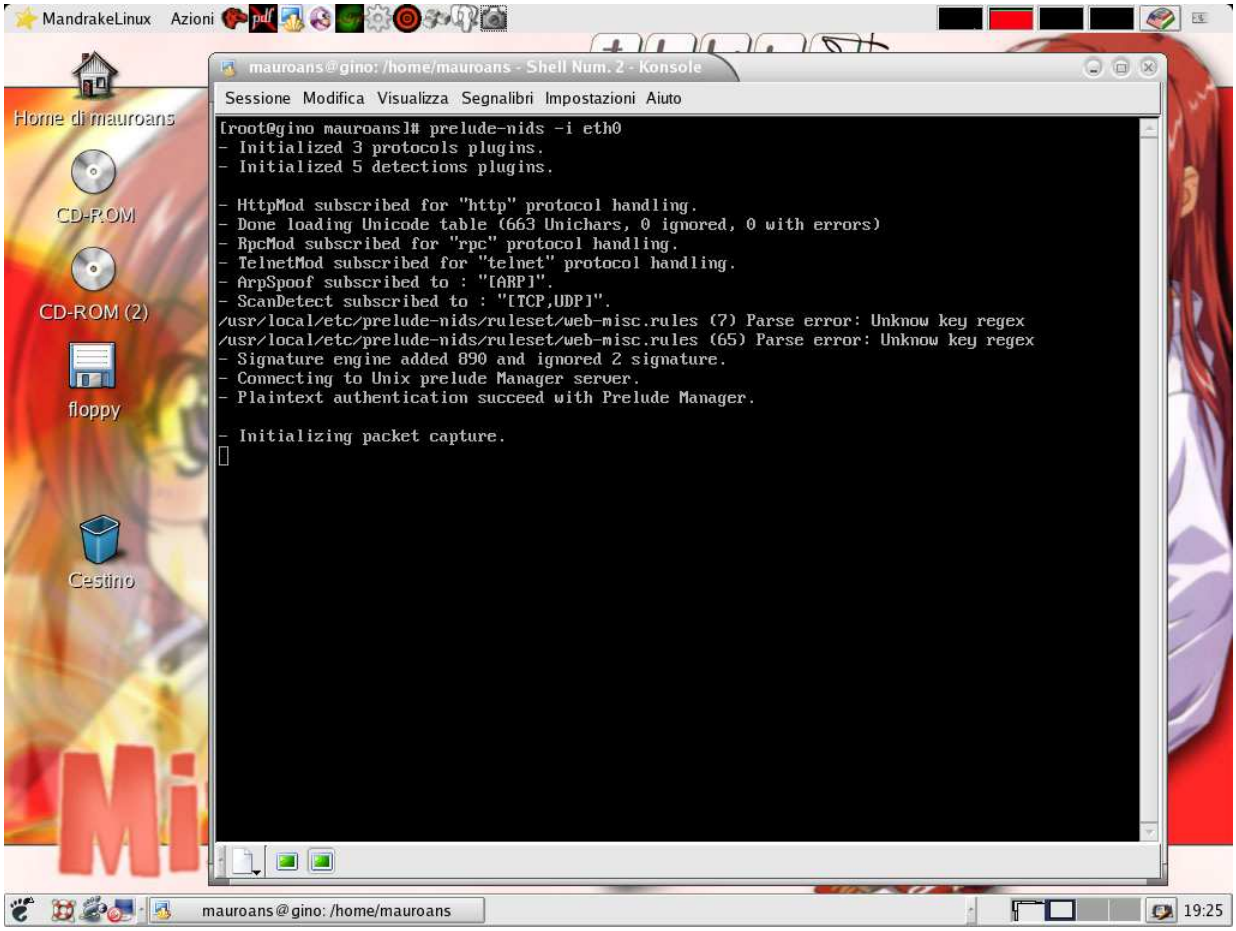

Figura 8.2: Attivazione di Prelude-NIDS

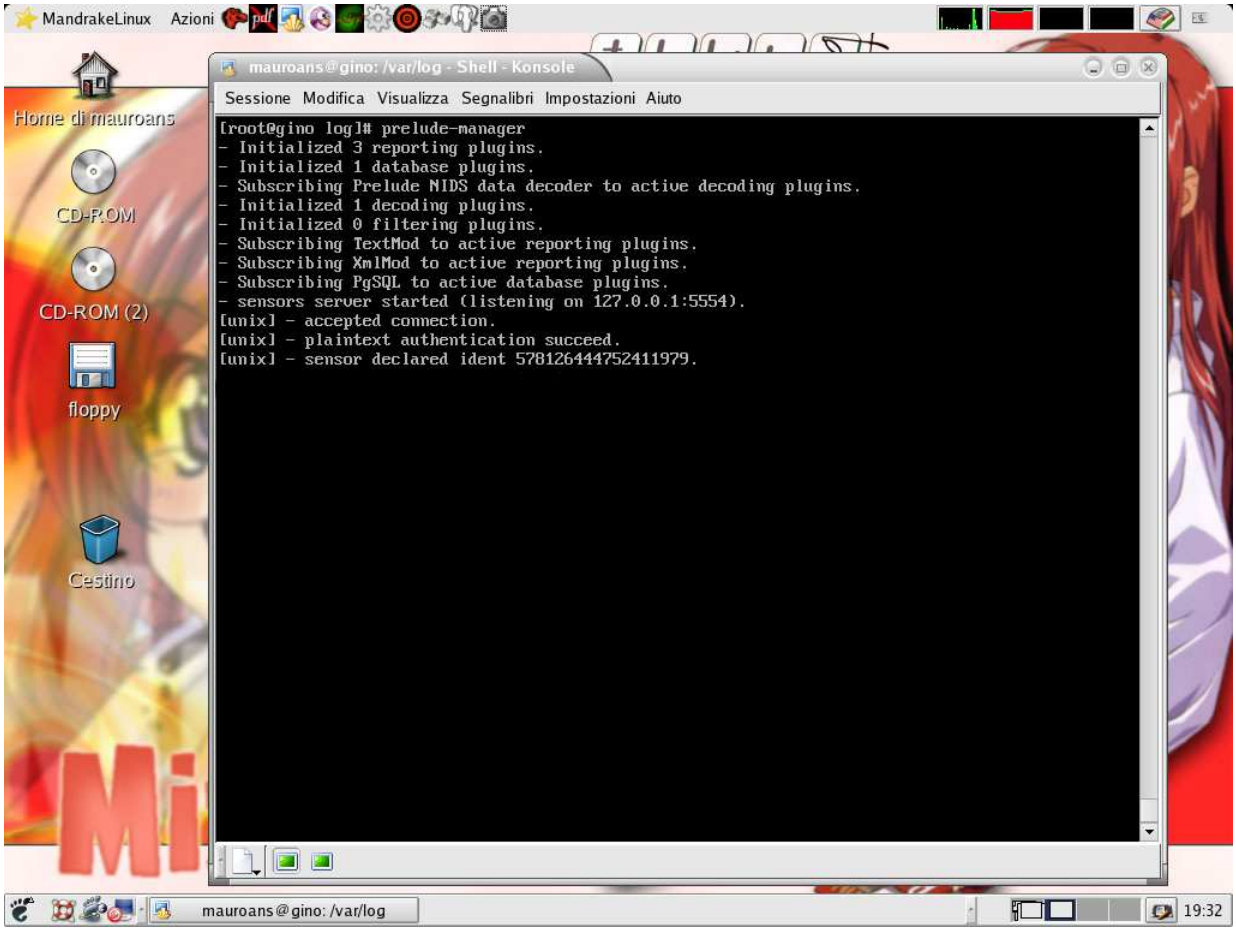

Figura 8.3: Connessione tra manager e sensore

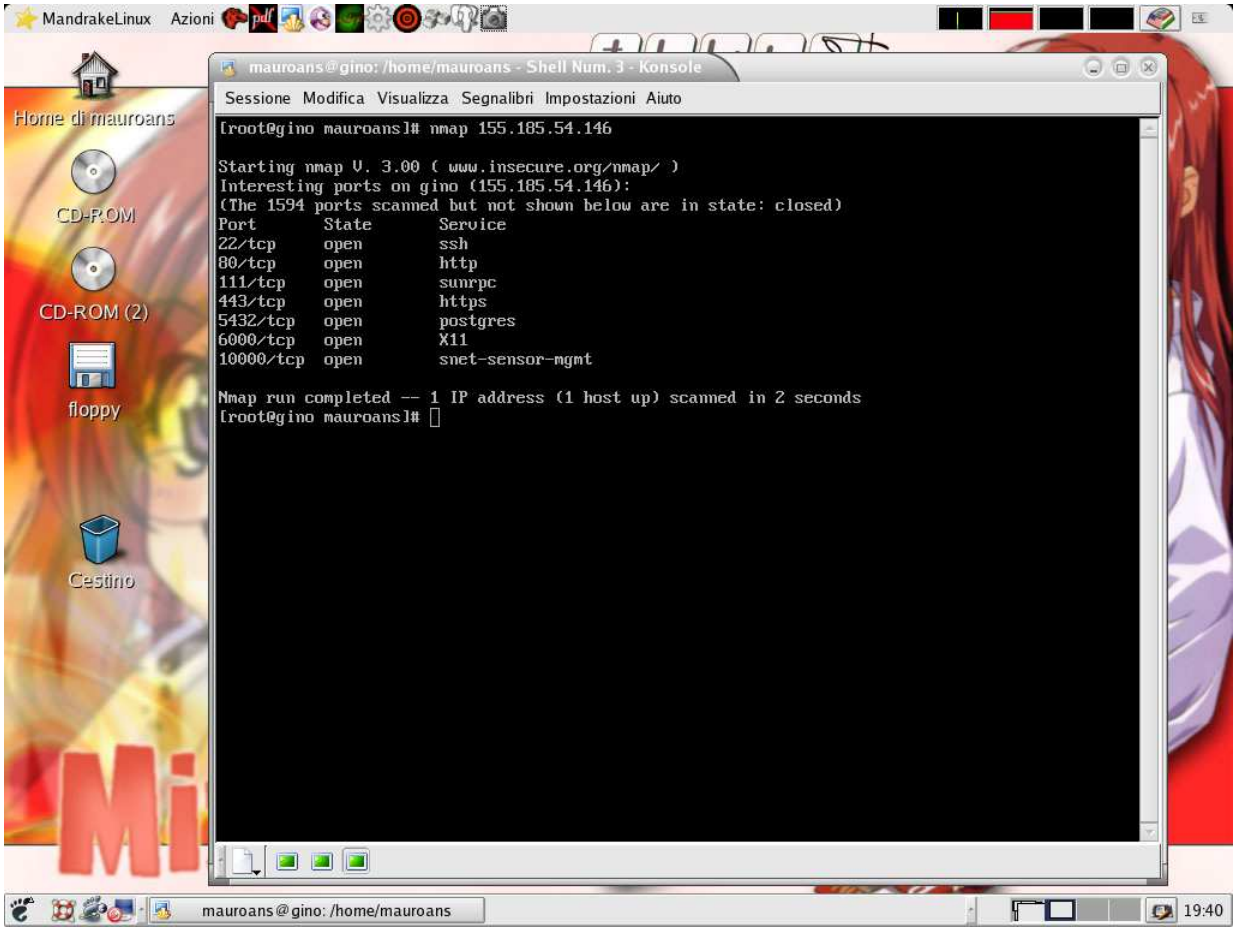

Figura 8.4: Attivazione di NMAP

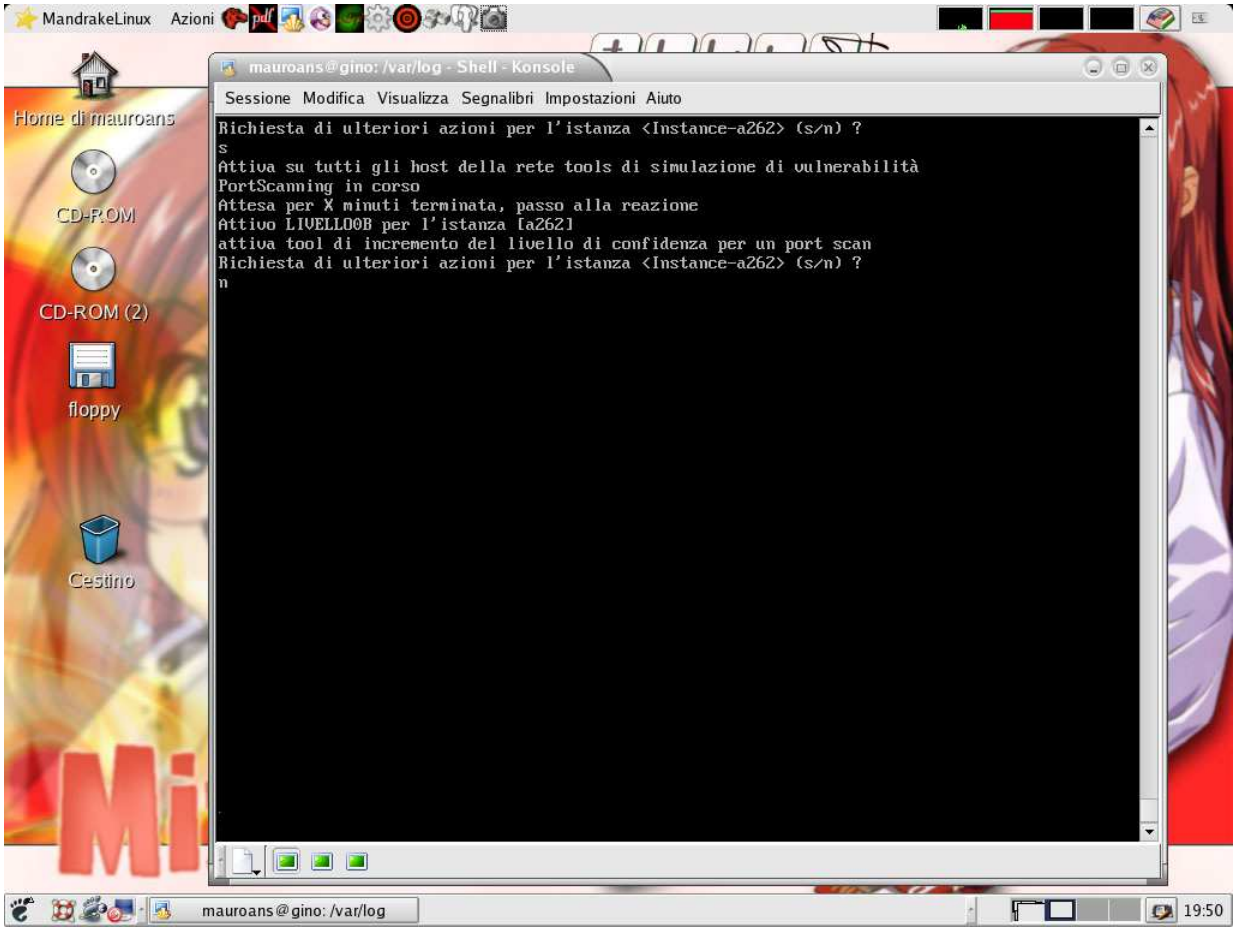

Figura 8.5: Riconoscimento dell'attacco da parte di SLUNP

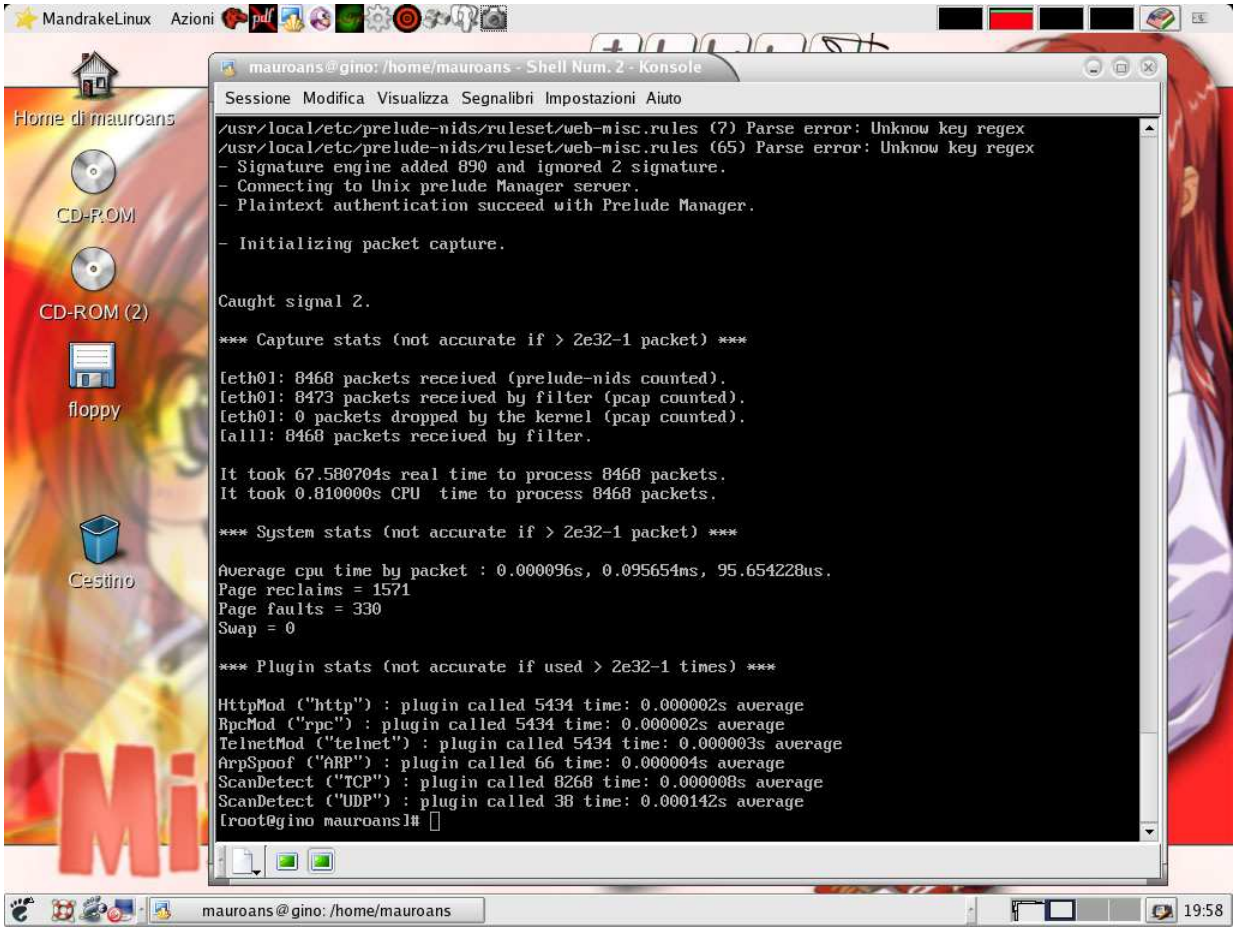

Figura 8.6: Disattivazione di Prelude-NIDS

Bibliografia

[ACQUIRE] Http://www.com/ai/

[ART*] Http://www.brightware.com/

[COMDALE] Http://www.comdale.com

[Rete-Alg] "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match problem", Charles L. Forgy, Artificial Intelligence 19 (1982), 17-37

[Compl-Rete] "Expert Systems: Principles and Programming", Giarratano and Riley, Second Edition, PWS Publishing (Boston 1993)

[FOCL] "The Role of Prior Knoledge in Inductive Learing", Pazzani M. and Klimber D., Machine Learning 9:54-97, 1992

[SOAR] Http://ai.eecs.umich.edu/soar/

[BABYLON] "The AI Workbench BABYLON"

[DYNACLIPS] Ftp.cs.cmu.edu:/user/ai/areas/expert/systems/clips/dyna/

[FUZZY] Http://ai.iit.nrc.ca/home_page.html

[AGENT-CLIPS] Ftp.cs.cmu.edu:/user/ai/areas/expert/systems/clips/agent

[ADS] Http://www.cs.cofc.edu/~manaris/ai-education-repository/expert-systems-tools.html

[wxCLIPS] Http://www.aiai.ed.ac.uk/~jacs/wxclips/wxclips.html

[MOBAL] Ftp.gmd.de:/gmd/mlt/Mobal/

[CPR] Http://www.haley.com/

[Easy] Http://www.haley.com/

[Exsys] Http://www.exsysinfo.com/

[CLIPS] Http://www.ghg.net/clips/CLIPS.html

[FLEX] Http://www.lpa.co.uk

[GBB] Http://www.bbtech.com/

[MailBot] Http://www.polaris.net/~daxtron/mailbot.htm

[HUGIN] Http://hugin.dk/

[ILOG] Http://www.ilog.fr

[Rete++] Http://www.haley.com/

[CIDF] Http://www.isi.edu/gost/cidf/draft/architecture.txt

[IDMEF] Http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt

[LIBPCAP] Http://www.tcpdump.org

[SNORT] Http://www.snort.org

[IDWG] Http://www.ietf.org/html.charters/idwg-charter.html

[UML] "UML Distilled Second Edition", Martin Fowler and Kendall Scott, Addison-Wesley Edition, 2000

[IANA] Http://www.iana.org/

[SNORT] Http://www.snort.org

[NMAP] Http://www.insecure.org/nmap/

[FIR] Http://www.benzedrine.cx/pf-paper.html

[OPS5] Http://burks.brighton.ac.uk/burks/foldoc/40/84.htm

[JESS] Http://herzberg.ca.sandia.gov/jess/

[MIKE] Http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?MIKE

[PROLOG] Http://www.csci.csusb.edu/dick/samples/prolog.syntax.html

[WINDEXS] Http://www.toupin.com/services/portfolio/samples/automation.htm

[MAC] Http://www.apple.com

[UNIX] Http://www.geek-girl.com/unix.html

[SUN] Http://www.sun.com/index.html

[Win] Http://www.microsoft.com

[RT-Expert] Http://www.cs.cofc.edu/~manaris/ai-education-repository/expert-systems-tools.html

[AKS] Http://www.angoss.com

[DClas] Http://www-2.cs.cmu.edu/Groups/AI/html/faqs/ai/expert/part1/faq-doc-7.html

[OPS83] Http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?OPS83