UNIVERSITÀ CA' FOSCARI – VENEZIA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica (Vecchio Ordinamento)

S-Flow

Semantic Web e XML per il desktop del futuro

Laureando: Christian Barbato (771459)

Relatore: Chiar.mo prof. Massimo Marchiori
Correlatore: Dott. Renzo Orsini

Anno Accademico 2004-2005

Indice

1 Introduzione **1**

2 Tecnologie utilizzate **5**
2.1 Semantic Web 5
2.1.1 RDF 10
2.1.2 OWL 19
2.1.3 SPARQL 22
2.1.4 Redland: librdf 25
2.2 Python 25
2.2.1 Piccola introduzione 26
2.3 wxWidgets 28

3 Il modello **29**
3.1 Metadati 29
3.2 Mimetype 29
3.2.1 Lo schema 30
3.3 Applicazioni 32
3.3.1 Lo schema 34
3.4 Operazioni 35
3.4.1 Parametri 36
3.4.2 Categorie 36
3.4.3 Livello di complessità 39
3.4.4 Gli schemi 39
3.4.5 Il problema delle operazioni copia 43
3.5 Il modello finale 44

3.5.1 Ulteriori vantaggi 45

4 I prototipi **47**
4.1 Sflow 47
4.1.1 Requisiti 47
4.1.2 Funzionamento 52
4.1.3 Le classi del modello 54
4.1.4 Le classi dell'interfaccia utente 62
4.1.5 Grafico UML 73
4.1.6 Adattamento di applicazioni esistenti al modello . . 73
4.1.7 Design dell'interfaccia 78
4.1.8 Esempi di flussi 82
4.1.9 Altre caratteristiche 88
4.1.10 Cross-platform 90
4.1.11 Problemi 91
4.2 Application Helper 94
4.2.1 Le classi del modello 96
4.2.2 Le classi dell'interfaccia utente 96

5 Manuale utente **99**
5.1 Cos'è Sflow 99
5.2 Cos'è AppHelp 101
5.3 Installazione in ambiente Windows 101
5.3.1 Integrazione con Windows 102
5.4 Installazione in ambiente MacOS X 103
5.4.1 MacOS X 10.3 (Panther) 104
5.4.2 MacOS X 10.4 (Tiger) 104
5.5 Installazione in ambiente GNU/Linux 104
5.5.1 Integrazione con Gnome 105
5.6 Il repository dei file RDF 106
5.7 Gli applicativi esterni 107
5.7.1 L'utilità di Py2Exe 112
5.8 Invocazione di Sflow 112
5.9 Utilizzo di Sflow 113
5.9.1 Scelta della complessità delle applicazioni 117

5.9.2 Applicativi multipli 117
5.10 Invocazione di AppHelp 118
5.11 Utilizzo di AppHelp 118

6 Conclusioni **120**
6.1 Risultati ottenuti 120
6.2 Possibili approfondimenti 120
6.2.1 Cercare l'applicazione corretta 120
6.2.2 Suddivisione del flusso 121
6.2.3 Iniziare il flusso da interrogazioni su metadata . . . 122
6.2.4 Integrazione con applicazioni remote 122
6.2.5 Andare oltre il mimetype 122

A Glossario **125**

Elenco delle figure

1.1 Menu contestuale *Apri con* di Windows 2
1.2 Pulsante di richiamo per Paint 3

2.1 Esempio di tripla RDF 13
2.2 Risorsa RDF fittizzia 16
2.3 Blank-Node RDF 17

3.1 Mime-type e Content-type 30
3.2 Mime-type e Applicazioni 33
3.3 Applicazioni e operazioni 37
3.4 Operazioni in dettaglio 38
3.5 Equivalenze tra operazioni 44
3.6 Esempio di flusso di operazioni 45

4.1 Sflow: Finestra di dialogo per mime-type ignoti 53
4.2 Sflow: "Apri con" intelligente 53
4.3 Sflow: esempio di operazione 54
4.4 Sflow: Operazioni in cascata 55
4.5 Diagramma UML delle classi di Sflow 74
4.6 Sflow: Operazione ridimensionamento immagine 77
4.7 Schema interfaccia utente 79
4.8 Schema interfaccia utente di SummaryPanel 79
4.9 Schema interfaccia utente di OperationPanel 81
4.10 Schema interfaccia utente di OutputPanel 81
4.11 Esempio di flusso a partire da un documento Pdf 83
4.12 Esempio di flusso a partire da varie immagini Jpeg 84
4.13 Esempio di flusso a partire da un documento Mp3 85

4.14 Sflow: scelta del grado di complessità 89
4.15 Sflow: scelta esplicita dell'applicativo 90
4.16 Sflow in ambiente Gnome 91
4.17 Sflow in ambiente MacOS X 92
4.18 Sflow in ambiente Windows 92
4.19 Schermata di AppHelp 95

5.1 Esempio di flusso in Sflow 100
5.2 Creazione link per Windows 102
5.3 Sflow integrato con Windows 103
5.4 Sflow integrato con Nautilus 106
5.5 Esempio di posizionamento della cartella contenente i documenti RDF 107
5.6 Sflow: pannello iniziale 113
5.7 Sflow: pannello operazione 114
5.8 Sflow: Particolare sulla lista delle categorie 114
5.9 Sflow: Operazione con parametri 115
5.10 Sflow: Particolare sul pannello dei nomi in output 115
5.11 Sflow: Esempio di flusso di lavoro 116
5.12 Sflow: Particolare della progress-bar 116
5.13 Sflow: complessità delle applicazioni 117
5.14 Sflow: applicativi multipli 118
5.15 Interfaccia di AppHelp 119

6.1 Possibile flusso di operazioni multiplo 121

Elenco delle tabelle

3.1 Classe OWL `Mimetype` 31
3.2 Classe OWL `Application` (prima parte) 34
3.3 Classe OWL `Category` 39
3.4 Classe OWL `Operation` 40
3.5 Classe OWL `Application` (seconda parte) 40

4.1 Requisisti funzionali 48
4.2 Requisisti non funzionali 50
4.3 Classe Mimetype 56
4.4 Classe MimetypeManager 56
4.5 Classe Category 59
4.6 Classe CategoriesManager 59
4.7 Classe Operation 60
4.8 Classe Parameter 60
4.9 Classe Application 61
4.10 Classe Flusso 62
4.11 Classe MainFrame 63
4.12 Classe OpScrolledWindow 65
4.13 Classe SummaryPanel 66
4.14 Classe OperationPanel 67
4.15 Classe TextCtrlValidator 69
4.16 Classe OutputPanel 70
4.17 Classe GaugePanel 71
4.18 Classe RemoveOperation 71
4.19 Classe ChangedOperation 72
4.20 Classe EditMimesDialog 72

4.21 Classe AppHelp 96
4.22 Classe AppHelpMainFrame 97
4.23 Classe ApplicationPanel 98

Capitolo 1

Introduzione

L'attuale modello di interfaccia utente nei computer desktop, e non solo, è fortemente basato sul concetto di applicazione. È ineffetti innegabile il ruolo principale che le applicazioni hanno nell'utilizzo del computer ma l'enorme mole di applicativi oggi disponibili e la loro scarsissima integrazione con i vari sistemi operativi, rende difficile l'approcio alla macchina da parte dell'utente comune. Ora come ora, infatti, molto spesso l'utente inesperto non ha idea di quale applicazione o, ancora peggio, di che genere di applicazione usare per poter raggiungere il proprio scopo.

Qualora, ad esempio, un utente alle prime armi volesse visualizzare un'immagine potrebbe usare un semplice e leggero image-viewer ma qualora volesse andarla a modificare quest'ultimo non sarebbe più sufficiente e dovrebbe utilizzare un image-editor, avendo inoltre cura di sceglierne uno con caratteristiche sufficienti da permettergli di effettuare le modifiche desiderate.

Prendendo in esame il caso concreto di trovarsi in un ambiente operativo Microsoft Windows, l'utente in questione si auspicherebbe che eseguendo un'operazione di doppio-click sull'icona rappresentante l'immagine quest'ultima fosse aperta e resa visibile da qualche tipo di programma. Le immagini però non sono tutte codificate con lo stesso formato e non tutti gli applicativi di visualizzazione di immagini sono in grado di leggere tutti i tipi di codifica. Vero è che i formati più comuni sono comunque generalmente gestiti dal sistema operativo stesso o da qualche applicazione

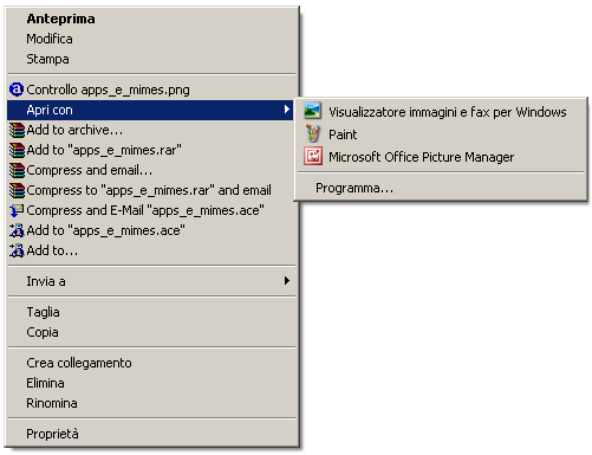

Figura 1.1: Menu contestuale *Apri con* di Windows

facente parte dell'ambiente operativo. Va però detto come, in ambiente Windows, un tipo di documento sia legato, di default, ad un'unica applicazione. Riprendendo infatti l'esempio dell'immagine, se l'utente volesse visualizzarla con un'applicativo diverso dovrebbe avere la fortuna che l'aplicazione di suo gradimento, durante la fase di installazione, abbia fatto in modo di apparire nel menu contestuale "Apri con" (vedi fig. 1.1) altrimenti l'utente dovrebbe aggiungerla a mano attraverso una procedura non semplice per l'utente alle prime armi. E tutto ciò solo per quel che riguarda Windows, altri sistemi operativi hanno altre problematiche ed altre modalità per risolverle.

Supponendo comunque che la funzione anteprima per le immagini delle ultime versioni di Windows sia sufficiente, pensiamo al problema che nasce qualora l'utente voglia andare a modificare l'immagine in questione. Nella versione XP di Windows questa problematica è stata risolta tramite un piccolo pulsante (vedi fig. 1.2) che va a richiamare l'applicazione di sistema `Paint`, la quale permette di fare delle semplici operazioni di editing. Ma se `Paint` non riuscisse a soddisfare le esigenze del cliente? Ci si ritroverebbe nella stessa problematica sopra-descritta. L'utente dovrebbe **conoscere quale altra applicazione utilizzare**, richiamarla esplicitamente ed essere in grado di utilizzarla, il che, il più delle volte, è

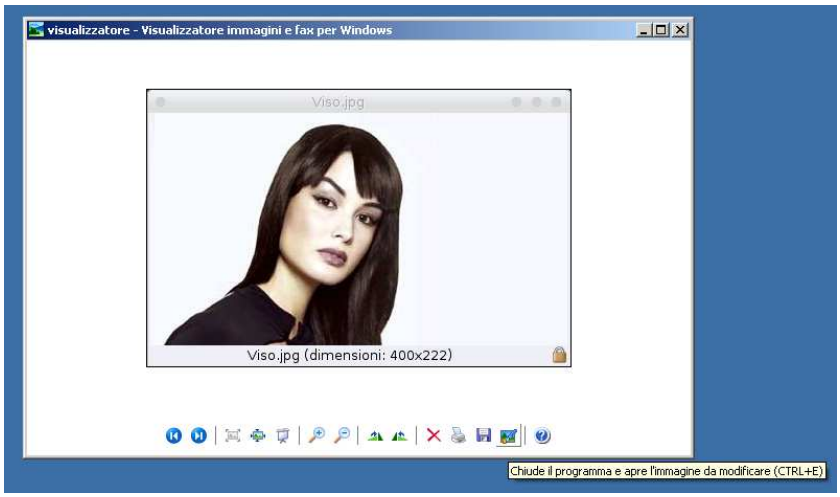

Figura 1.2: Pulsante di richiamo per Paint

tutt'altro che vero. E, ancora una volta, tutto ciò vale solo per l'ultima versione del sistema operativo Microsoft ma non per altre versioni o addirittura per altri sistemi operativi. Non esiste cioè un sistema standard di associazione tra mime-type ed applicativi e il **concetto di applicazione** risulta essere di difficile comprensione da parte dell'utente comune.

Tutto ciò in genere suscita confusione e frustrazione, se poi si considera la presenza di decine di applicazioni equivalenti con differenti tipologie di licenza (freeware, open-source, shareware, commerciali) risulta evidente che un utente alle prime armi riuscirà con molta difficoltà a districarsi tra le possibili scelte e a completare l'operazione desiderata.

Sarebbe molto più semplice se l'utente potesse concentrarsi solamente sul documento che deve stilare o trattare e sulle modifiche che desidera applicare lasciando al sistema il compito di fornire la soluzione adatta. L'applicazione dovrebbe essere un entità il più possibile trasparente e astratta, mentre dovrebbero avere grande visibilità le operazioni effettuabili sui documenti arrivando quindi ad un sistema che si potrebbe definire come "***operation-oriented***" al posto dell'attuale "***application-oriented***".

La seguente tesi si propone quindi di illustrare un modello concettuale (vedi capitolo 3) e dei prototipi funzionali (vedi capitolo 4) al fine di

dimostrare un nuovo approcio alla manipolazione dei documenti e alla creazione di veri e propri *flussi di lavoro* nei computer desktop.

Il modello permetterà di descrivere attraverso dei vocabolari di metadati le possibili relazioni tra mime-type ed applicazioni e, soprattutto, fornirà un metodo attraverso il quale gli applicativi potranno segnalare le funzioni che sanno svolgere ed il modo in cui possono essere interrogate al fine di compiere effettivamente tali operazioni. Fornendo inoltre informazioni riguardanti i documenti in uscita alle singole operazioni (il loro mime-type) sarà possibile connetterle una di seguito all'altra formando delle vere e proprie *pipeline* di operazioni che **vanno al di là del concetto di applicazione** di fatto nascondendole all'utente.

Il tutto avviene utilizzando protocolli standard e librerie cross-platform e open-source andando quindi oltre le particolarità e limitazioni dei singoli sistemi operativi.

A tale scopo verranno utilizzate recenti tecnologie inerenti la manipolazione di dati e metadati nate nell'ambito del progetto sul Semantic Web come XML [24][8], RDF [10], OWL [16], il linguaggio di interrogazione su basi di dati semantici SPARQL [9]. Il linguaggio di programmazione utilizzato sarà Python [32] [18] con varie librerie a contorno (vedi capitolo 2).

Capitolo 2

Tecnologie utilizzate

2.1 Semantic Web

In un articolo molto noto, pubblicato nel maggio del 2001 sulla prestigiosa rivista Scientific American, Tim Berners-Lee (accreditato come uno degli inventori dell'attuale World Wide Web), James Hendler e Ora Lassila scrivono: "*Il Semantic Web è un'estensione dell'attuale Web, nella quale all'informazione viene dato un significato ben definito, permettendo così ai computer e alle persone di lavorare meglio in cooperazione.*" [22]

Come dice la citazione riportata poco sopra, il Semantic Web non è una nuova entità indipendente o alternativa rispetto al Web come lo conosciamo oggi, ma è piuttosto una sua estensione che ne dovrebbe aumentare enormemente le potenzialita. Eppure il Web attuale sembra essere uno strumento già estremamente potente. In che senso dunque se ne vogliono accrescere le potenzialità? Un modo semplice per rispondere a questa domanda è quello di riflettere su alcune limitazioni del Web con cui la maggior parte dei suoi utenti si scontra molto spesso. Eccone alcuni esempi:

Ricerca di documenti Tutti ormai usiamo il Web per trovare documenti utili alla nostra vita professionale o personale. Per far ciò, sono disponibili due strade: seguire i link da una pagina all'altra fino a trovare quello che cercavamo, o servirci del nostro motore di

ricerca preferito (Google, Yahoo, Altavista, Virgilio, etc.) per ottenere una lista di link tra i quali scegliere quelli di nostro interesse. Ognuno dei due metodi ha vantaggi e svantaggi: seguire i link ipertestuali è un processo cognitivamente molto ricco per noi umani, che in genere capiamo il contenuto del documento a cui punta il link dalla descrizione che ne troviamo nella pagina da cui partiamo, e da altra informazione di contesto (per esempio, se troviamo un link descritto come "Pinco Pallino" nella pagina del Dipartimento di Informatica dell'Universita di Venezia, sezione "People", capiamo non solo che cliccando sul link troveremo la pagina di un tale che si chiama Pinco Pallino, ma anche che egli con ogni probabilita è un membro del Dipartimento di Informatica di Venezia e non un calciatore del Trento). Tuttavia, tale processo può essere estremamente dispendioso in termini di tempo, e difficile da avviare nei casi in cui non sappiamo da dove partire per iniziare a seguire link che ci possano condurre a ciò che cerchiamo (per esempio, da dove partiamo a seguire link se cerchiamo informazione su un certo Pinco Pallino, del quale però non sappiamo nient'altro se non che raccontava barzellette interessanti sulla spiaggia?).
Usare motori di ricerca ha l'evidente vantaggio di richiedere pochissima informazione in partenza (dal nome "Pinco Pallino" possiamo trovare molte pagine che contengono tale sequenza di caratteri). Tuttavia, anche questo metodo ha dei limiti noti a tutti. Innanzitutto, i motori di ricerca non coprono tutto il contenuto del Web (si parla di un 80% di "hidden web", cioè contenuto non indicizzato o persino non indicizzabile per ragioni tecniche). Ma soprattutto la ricerca per parole chiave può essere frustrante per l'elevato numero di "falsi positivi" (pagine che contengono la parola chiave da noi inserita, ma non parlano di quello che cercavamo, per esempio per problemi di polisemia) e di "falsi negativi" (nei risultati della ricerca non compaiono pagine che sarebbero state di nostro interesse, ma che non contenevano esattamente la parola chiave da noi immessa e magari contenevano un suo sinonimo).

Ricerca di informazione Un altro forte limite del Web attuale è che

la ricerca, comunque sia effettuata, restituisce sempre documenti, e non esattamente l'informazione che stiamo cercando. Per cui, se vogliamo sapere quali articoli ha pubblicato Pinco Pallino nel 2003 o quelli in cui ha avuto dei co-autori, i due metodi di ricerca descritti sopra ci permetteranno di trovare la pagina delle pubblicazioni di Pinco Pallino (se siamo fortunati), ma dopo ci sarà richiesto un certo lavoro manuale per creare la lista che ci interessa. Al momento attuale, nulla ci supporta in richieste come quelle descritte. Per esempio, non c'è modo di fare riferimento al concetto di "co-autore", o di "anno di pubblicazione" di un articolo. Tali ricerche possono solo essere fatte a mano o approssimate combinando piu parole chiave, ma questo ci espone a tutti i problemi descritti sopra.

Integrazione di informazione Spesso diverse parti dell'informazione che ci interessa sono contenute in diversi documenti, magari disponibili su siti diversi. Per esempio, se volessimo sapere quali autobus (o traghetti) sono disponibili dall'aeroporto di Venezia verso il centro città in coincidenza con un certo volo, dovremmo consultare separatamente il sito della compagnia aerea per sapere gli orari dei voli, poi quello della compagnia di trasporto terrestre per gli orari dei bus, e infine combinare manualmente le due cose in un piano di viaggio. Ancora una volta, nulla ci permette di descrivere concetti come "prima", "dopo", "coincidenza", "percorso" e così via, né di combinare informazione proveniente da fonti diverse in un piano che risolva il nostro problema.

Cooperazione Infine, è chiaro che il Web attuale non permette la cooperazione tra programmi e tra programmi e utenti umani per risolvere problemi complessi. La maggior parte dei siti web non sono progettati per fornire servizi ad altri servizi, ma semplicemente come contenitori di informazioni che possono essere estratte a richiesta. Per contro, molte applicazioni (per esempio, l'organizzazione di un viaggio) richiederebbero da un lato che i siti Web di varie organizzazioni potessero interagire in modo flessibile e dinamico (per esempio, per comporre un piano di viaggio e alloggio e un programma di visite turistiche integrate con il piano di viaggio), e dall'altro che

gli utenti potessero intervenire nel processo interagendo con i programmi dei vari siti (per esempio, verificare interattivamente che il risultato sia conforme alle esigenze del potenziale viaggiatore, permettendogli di richiedere modifiche o proposte alternative nel corso del processo di definizione).

A questo tipo di problemi intende rispondere il Semantic Web.

Il Semantic Web è quindi un progetto di ricerca basato su un programma molto preciso: quello di ridefinire e ristrutturare i dati su Web in modo che il loro significato sia accessibile non solo a utenti umani (come avviene per il Web attuale), ma anche e soprattutto a programmi che li utilizzano per manipolarli, integrarli, renderli disponibili per diverse applicazioni, non solo mostrarli, come succede ora. Tale programma si basa su tre pilastri fondamentali:

Rappresentazione della conoscenza Per poter dare un risultato veramente soddisfacente per l'utente, i programmi scritti per il Semantic Web devono avere un qualche tipo di accesso al significato dei dati con cui lavorano. Per esempio, a differenza degli attuali motori di ricerca, devono poter capire se "Pinco Pallino" è una persona o una marca di profumo, e cosa deriva dalla decisione tra uno dei due significati. Pertanto, l'aggettivo "semantico" nella locuzione "Semantic Web" traduce questa volontà di creare qualcosa che sia significativo per dei programmi di computer, e non solo per l'utente umano. Il primo passo per mettere le macchine in grado di operare in maniera più efficiente sono dei documenti ben strutturati corredati da metadati che esprimono parte di quello che c'è da sapere su un certo insieme di dati, e ciò che da tale conoscenza è possibile inferire mediante regole di ragionamento. Questo significa che il Semantic Web intende recuperare tutta la tradizione della ricerca in Intelligenza Artificiale che va sotto il nome di rappresentazione della conoscenza, focalizzandola tuttavia su un nuovo obiettivo molto ben delimitato e preciso. Lo strumento tecnico elaborato per esprimere i metadati nel Semantic Web sono dei linguaggi di annotazione (o markup language) costruiti a partire da XML [24], come per esempio il noto RDF (Resource Description Framework, vedi

sez. 2.1.1) [10]. Le annotazioni servono a rappresentare il significato dei dati annotati. I linguaggi di annotazione ci permettono quindi di annotare la stringa "Pinco Pallino" con l'informazione che Pinco Pallino lavora presso il Dipartimento di Informatica dell'Universita di Venezia, che collabora ai progetti X, Y e Z e che insegna il corso di Programmazione.

Utilizzo di ontologie Tuttavia, l'introduzione di linguaggi di annotazione sono solo il primo passo verso l'ingresso della semantica nel Web, poichè ai linguaggi da soli manca ancora uno strato concettuale che leghi i termini tra loro, mostrandone le relazioni e facendone quindi emergere il significato. Questo compito è svolto dalle cosiddette ontologie, cioè documenti o file che hanno lo scopo di esprimere il significato di un certo insieme di termini definendone le relazioni reciproche. Per dare un'intuizione di che cosa un'ontologia aggiunga a un linguaggio di annotazione, si può pensare alla differenza esistente tra un semplice elenco di parole italiane e un dizionario concettuale. Il primo ci dice solo quali parole è lecito usare, il secondo fornisce anche un insieme di relazioni logico-grammaticali con altri termini (per esempio, che ci dica che Pinco è un professore, che professore è un ruolo accademico e sociale, che ogni professore afferisce a un Dipartimento accademico, che i Dipartimenti sono strutture organizzative delle Università, che l'Universita è un tipo di istituzione, ecc.). Le ontologie pensate per il Web hanno spesso la forma di tassonomie corredate di un insieme di regole di inferenza, e forniscono definizioni formali delle relazioni tra termini (vedi OWL sez. 2.1.2).

Agenti software Gli agenti software sono programmi che dovrebbero sfruttare la conoscenza contenuta nei metadati per fornire servizi complessi agli utenti del Semantic Web. Per far ciò, gli agenti devono essere dotati di un buon livello di autonomia (non possono rivolgersi continuamente all'utente per decidere cosa fare) e di capacità di ragionare in modo complesso sull'informazione a cui hanno accesso. Questo significa al minimo essere in grado di rappresentare gli obiettivi di un certo utente, di mettere in atto una sequenza di

azioni che possa soddisfare tali obiettivi, ed eventualmente di cooperare con altri agenti per ottenere tale risultato. Naturalmente, ognuno dei punti elencati comporta l'emergere di molti problemi, alcuni dei quali di difficile soluzione. Il più generale di tutti ha a che fare con un concetto solo apparentemente tecnico, vale a dire il concetto di interoperabilità. Il che significa non solo l'abilità di programmi progettati indipendentemente e implementati su piattaforme diverse di scambiarsi dati (cosa che già oggi in parte avviene), ma soprattutto di cooperare tra di loro sulla base di una "comprensione" del significato dei dati oggetto di scambio (in questo senso si parla di interoperabilità semantica). In altre parole, ciò richiede che i programmi per il Semantic Web (e in particolare gli agenti software) debbano "capirsi" senza presupporre un accordo a priori su cosa significhi un certo dato (detto altrimenti, due agenti diversi potrebbero non condividere la rappresentazione del mondo su cui agiscono), e su quale uso verrà fatto di un certo dato.
Inoltre, agenti molto evoluti dovrebbero anche essere capaci di dedurre, a partire dall'analisi di un documento e grazie a processi inferenziali, nuovi metadati non presenti in esso, di acquisire nuove "conoscenze e capacità" quando vengono in contatto con nuove ontologie e di ricavare informazioni importanti anche da servizi che rispondano solo parzialmente alle richieste dell'agente stesso. In aggiunta, ci si aspetta che la cooperazione tra questi agenti avvenga anche in base a una capacita di ragionare sulle intenzioni degli utenti umani e degli altri agenti software, e nel rispetto di norme di tipo etico e legale.

2.1.1 RDF

Va sottolineato come sia difficile automatizzare il Web restando ancorati alla sua architettura originaria, in cui tutte le informazioni sono *machine-readable*, ma non *machine-understandable* (ossia le informazioni sono processabili dalle macchine ma non "capite"), e come la soluzione al problema possa venire dai metadati. L' uso efficace dei metadati, tuttavia, richiede che vengano stabilite delle convenzioni per la semantica,

la sintassi e la struttura. Le singole comunità interessate alla descrizione delle loro risorse specifiche definiscono la semantica dei metadati pertinenti alle loro esigenze. La sintassi, cioè l' organizzazione sistematica dei `data element` per l' elaborazione automatica, facilita lo scambio e l' utilizzo dei metadati tra applicazioni diverse. La struttura può essere vista come un vincolo formale sulla sintassi, per una rappresentazione consistente della semantica.

RDF (Resource Description Framework) [10] [4] [20] [19] è lo strumento base per la codifica, lo scambio e il riutilizzo di metadati strutturati, e consente l' interoperabilità tra applicazioni che si scambiano sul Web informazioni machine-understandable. I settori nei quali RDF può essere utilizzato e portare vantaggi sono i più disparati, basti citare, a titolo di esempio: descrizione del contenuto di un sito Web, o di una pagina, o di una biblioteca digitale; implementazione di intelligent software agent, per lo scambio di conoscenza e un utilizzo migliore delle risorse Web; classificazione del contenuto, per applicare criteri di selezione; descrizione di un insieme di pagine, che rappresentano un singolo documento logico; stabilire i criteri di proprietà intellettuale delle singole pagine; esprimere criteri di privacy preference degli utenti e le privacy policies di un sito Web; con il meccanismo della digital signature, contribuire alla creazione del Web of Trust, per le applicazioni nel commercio elettronico, la cooperazione, etc.. Il Resource Description Framework (RDF), quindi, non descrive la semantica, ma fornisce una base comune per poterla esprimere, permettendo di definire la semantica dei tag XML.

RDF è costituito da due componenti:

RDF Model and Syntax definisce il data model RDF e la sua codifica XML;

RDF Schema permette di definire specifici vocabolari per i metadati.

RDF Data Model

RDF fornisce un modello per descrivere le risorse. Come descritto precedentemente, le risorse hanno delle proprietà (o anche attributi o caratteristiche). RDF definisce una risorsa come un qualsiasi oggetto che

sia identificabile univocamente mediante un Uniform Resource Identifier (URI). Il data model RDF è molto semplice, ed è basato su tre tipi di oggetti:

Resources Qualunque cosa descritta da una espressione RDF viene detta risorsa (resource). Una risorsa può essere una pagina Web, o una sua parte, o un elemento XML all' interno del documento sorgente. Una risorsa può anche essere un' intera collezione di pagine web, o anche un oggetto non direttamente accessibile via Web (per es. un libro, un dipinto, etc.). Le risorse sono sempre individuate da un URI, eventualmente con un anchor id.

Properties Una property (proprietà) è un aspetto specifico, una caratteristica, un attributo, o una relazione utilizzata per descrivere una risorsa. Ogni proprietà ha un significato specifico, definisce i valori ammissibili, i tipi di risorse che può descrivere, e le sue relazioni con altre proprietà. Le proprietà associate alle risorse sono identificate da un nome, e assumono dei valori.

Statements Una risorsa, con una proprietà distinta da un nome, e un valore della proprietà per la specifica risorsa, costituisce un RDF statement. Uno statement è quindi una tupla composta da un soggetto (risorsa), un predicato (proprietà) e un oggetto (valore). L' oggetto di uno statement (cioè il property value) può essere un' espressione (sequenza di caratteri o qualche altro tipo primitivo definito da XML) oppure un' altra risorsa.

L'utilizzo di RDF può essere chiarito con qualche semplice esempio. Consideriamo queste due espressioni:

1. "Mario Rossi è l'autore del DocumentoX"
2. "L'autore del DocumentoX è Mario Rossi"

Ovviamente, le due espressioni sono del tutto equivalenti per un essere umano, in quanto veicolano la stessa informazione, mentre verrebbero viste come due espressioni diverse da una macchina. RDF, mediante il suo semplice modello basato su resource, property e value, intende fornire

un metodo non ambiguo per esprimere la semantica con una codifica comprensibile dalla macchina. Dato che RDF fornisce un meccanismo per associare le proprietà alle risorse, per poter dare un valore alle proprietà occorre che sia dichiarata la risorsa. Quindi, per prima cosa va dichiarata una risorsa che rappresenti il DocumentoX. Abbiamo quindi la tripla:

Resource	http://www.pincopanco.it/Mario/DocX
Property	author
Value	Mario Rossi

Lo statement dell'esempio verrebbe quindi rappresentato come:

http://www.pincopanco.it/Mario/DocX has Author Mario Rossi

e verrebbe rappresentata graficamente come in figura 2.1.

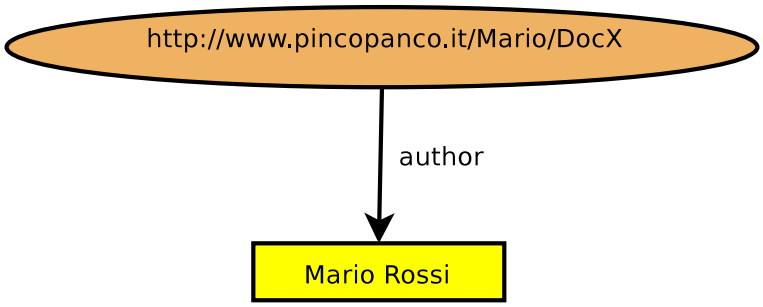

Figura 2.1: Esempio di tripla RDF

Containers

Talvolta, è necessario far riferimento a più di una risorsa, per esempio per descrivere il fatto che un libro è stato scritto da più autori, oppure che un documento è composto da una serie di componenti, oppure che una funzione può essere svolta da una delle persone elencate. RDF definisce tre tipi di contenitori (container):

Bag È una lista non ordinata di risorse o costanti. Viene utilizzato per dichiarare che una proprietà ha valori multipli, senza alcun significato particolare attribuito al loro ordine (per esempio, i componenti di una commissione). Sono ammessi valori duplicati.

Sequence È una lista ordinata di risorse o costanti. Viene utilizzato per dichiarare che una proprietà ha valori multipli, e che il loro ordine è significativo (per esempio, gli autori di un libro, un insieme di nomi di cui si voglia preservare l' ordine alfabetico). Sono ammessi valori duplicati.

Alternative È una lista di risorse o costanti che rappresentano una alternativa per il valore (singolo) di una proprietà. Può essere utilizzato, per esempio, per fornire titoli alternativi in varie lingue. È possibile definire proprietà sia dell' intero container che dei singoli elementi.

Namespaces

RDF consente alle singole comunità di definire la semantica. Tuttavia, non è possibile affidare la semantica semplicemente al nome, che potrebbe avere significati più o meno ampi a seconda degli interessi specifici delle singole comunità. RDF identifica univocamente le proprietà mediante il meccanismo dei namespace [23]. I namespace XML forniscono un metodo per identificare in maniera non ambigua la semantica e le convenzioni che regolano l'utilizzo delle proprietà identificando l'authority che gestisce il vocabolario. Uno degli esempi più noti è la Dublin Core Initiative [27], [26] che definisce, per esempio, il campo Subject and Keywords nel seguente modo:

```
Name: Subject and Keywords
   Identifier: Subject
   Definition: The topic of the content of the resource.
   Comment: Typically, a Subject will be expressed as keywords,
            key phrases or classification codes that describe a
            topic of the resource.
            Recommended best practice is to select a value from
            a controlled vocabulary or formal classification
```

scheme.

Si può quindi utilizzare un namespace XML per identificare in maniera non ambigua lo schema per il vocabolario Dublin Core puntando alla risorsa Dublin Core che ne definisce la semantica. Quindi, la descrizione di un sito Web mediante le proprietà definite nel vocabolario Dublin Core potrebbe essere:

```
<rdf:RDF
   xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
   xmlns:dc=http://purl.org/metadata/dublin_core#>

   <rdf:Description about=http://www.dlib.org>
      <dc:Title>
         D-Lib Program - Research in Digital Libraries
      </dc:Title>
      <dc:Description>
         The D-Lib program supports the community of people with
         research interests in digital libraries and electronic
         publishing.
      </dc:Description>
      <dc:Publisher>
         Corporation For National Research Initiatives
      </dc:Publisher>
      <dc:Date>
         1995-01-07
      </dc:Date>
      <dc:Subject>
         <rdf:Bag>
            <rdf:li>Research; statistical methods</rdf:li>
            <rdf:li>Education, research, related topics</rdf:li>
            <rdf:li>Library use Studies</rdf:li>
         </rdf:Bag>
      </dc:Subject>
      <dc:Type>World Wide Web Home Page</dc:Type>
      <dc:Format>text/html</dc:Format>
      <dc:Language>en</dc:Language>
   </rdf:Description>
</rdf:RDF>
```

Blank nodes

Sarebbe molto semplice se tutte le informazioni descrivibili con RDF fossero nelle forma di una tripla come visto in figura 2.1. Nei casi reali risulta utile poter disporre di meccanismi un po' più complessi. Si pensi, ad esempio, di dover descrivere l'indirizzo di una persona:
`http://example.com/MarioRossi haIndirizzo: 'Via dell'Informatica 30, Venezia, Italia'`
si potrebbe memorizzare l'intero indirizzo come un'unica risorsa ma qualora fosse richiesto di memorizzare separatamente via, numero civico, città e stato? In questo caso la risorsa rappresentante l'indirizzo andrebbe spezzata nelle varie parti. Si dovrebbe quindi creare una risorsa fittizzia come illustrato in figura 2.2.

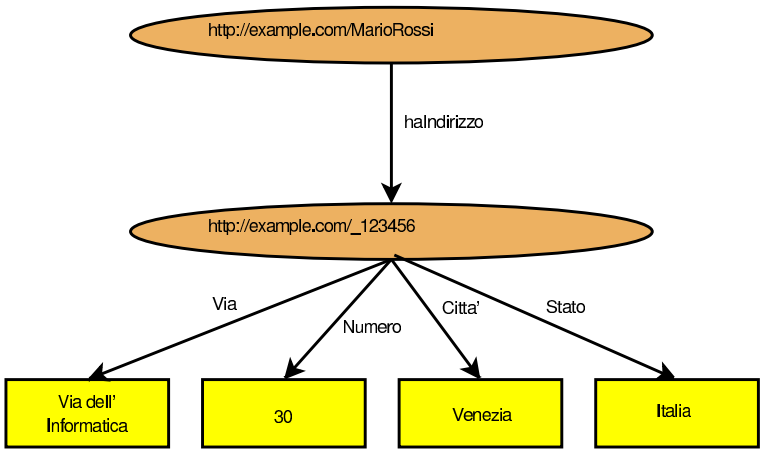

Figura 2.2: Risorsa RDF fittizzia

Questo modo di rappresentare le risorse può comportare la creazione di numerose risorse "intermedie". Di fatto però queste risorse non verranno mai utilizzate direttamente ma servono solo da ponte per arrivare alle informazioni vere e proprie. Per questo motivo vengono chiamate *blank-nodes* e graficamente vengono rappresentate come delle risorse "vuote", vedi figura 2.3.
Di fatto, quindi, i blank-node sono delle risorse sprovviste di URI.

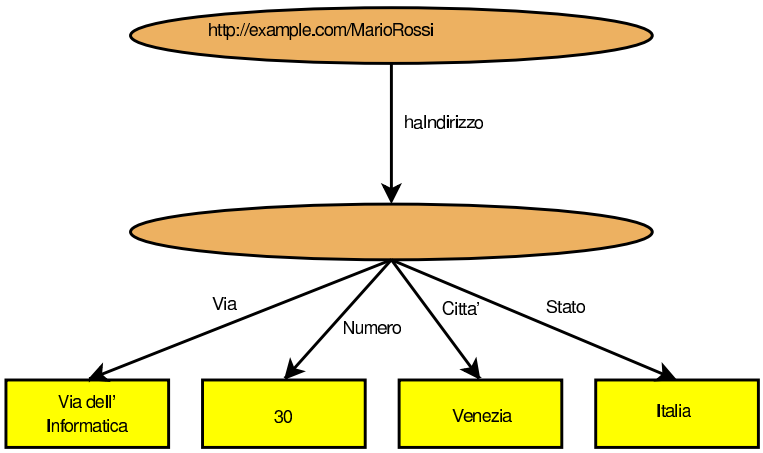

Figura 2.3: Blank-Node RDF

RDF Schema

Il data model RDF permette di definire un modello semplice per descrivere le relazioni tra le risorse, in termini di proprietà identificate da un nome e relativi valori. Tuttavia, RDF data model non fornisce nessun meccanismo per dichiarare queste proprietà, né per definire le relazioni tra queste proprietà ed altre risorse. RDF Schema permette di definire dei vocabolari, quindi l'insieme delle proprietà semantiche individuata da una particolare comunità. RDF Schema [5] permette di definire significato, caratteristiche e relazioni di un insieme di proprietà, assieme ad eventuali vincoli sul dominio e sui valori delle singole proprietà. Inoltre, implementando il concetto (transitivo) di classe e sottoclasse, consente di definire gerarchie di classi, con il conseguente vantaggio che agenti software intelligenti possono utilizzare queste relazioni per svolgere i loro compiti.

Sintassi per RDF

Come descritto precedentemente, il modello concettuale alla base di RDF è il grafo. Di certo, comunque, è necessaria una rappresentazione testuale dell'informazione espressa graficamente. A tal fine sono disponibili vari tipi di linguaggi. Il inguaggio ufficiale, RDF/XML [1], è un diretto derivato di XML ma a causa della sua "pesantezza" visiva, fonte di confusione per la sua scrit-

tura e lettura da parte dell'uomo, sono stati sviluppate altre forme espressive, quale, ad esempio, N3 [3] che sarà descritto qui di seguito.

RDF/XML

L'idea di fondo riguardo alla sintassi RDF/XML [1] può essere illustrata con un esempio. Si debba descrivere la seguente tripla:
http://www.pincopanco.it/index.html `ha una` **data-di-creazione** `pari a` **1 Settembre 2005**
allora, utilizzando RDF/XML si avrà:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:pp="http://www.pincopanco.it/termini/">

  <rdf:Description rdf:about="http://www.pincopanco.it/index.html">
      <pp:data-di-creazione>1 Settembre 2005</pp:data-di-creazione>
  </rdf:Description>

</rdf:RDF>
```

Il documento inizia quindi con la classica dichiarazione XML `<?xml version=1.0?>` che indica che il documento è redatto utilizzando il linguaggio XML versione 1.
La seconda linea inizia con `<rdf:RDF` il che indica che il documento andrà a rappresentare dati RDF. La riga continua indicando un *namespace* ossia un prefisso. In sostanza si dice che tutti i termini che iniziano con la dicitura `rdf:` fanno parte del namespace identificato dalla URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.
Allo stesso modo la terza linea dichiara un altro namespace.
Le linee da 4 a 6 descrivono la tripla richiesta. Lo si fa iniziando un tag `Description` con proprietà `about` uguale al soggetto della tripla in questione. Si procede quindi descrivendo le proprietà e si conclude chiudendo il tag `Description`.
Il documento finisce con la chiusura del tag rdf:RDF.

N3

In RDF, l'informazione è semplicemente un insieme di dichiarazioni, ognuna con un soggetto, verbo e oggetto, nient altro. In N3 [3] la tripla RDF può essere scritta esattamente così, con l'aggiunta di un punto finale:
`<#Mario> <#conosce> <#Luigi> .`

Ogni cosa, sia essa il soggeto, il verbo o l'oggetto, viene identificato con una URI ossia qualcosa come:
`<http://www.w3.org/2000/10/swap/test/s1.n3#includes>`
ossia una stringa composta dal *namespace*, o prefisso, e dal termine in questione. L'unica eccezione è che l'oggetto di una tripla può essere un `literal` ossia una stringa qualsiasi, un numero o un altro tipo di dato diverso da una URI.

Quando esistono diverse dichiarazioni in relazione allo stesso soggetto si può fare ricorso a due accorgimenti: un punto e virgola ; introduce un'altra proprietà dello stesso soggetto, e una virgola introduce un altro oggetto con lo stesso predicato e soggetto. Ad esempio:

```
<#Mario> <#figlio>  <#Stefano>, <#Luca>, <#Paola> ;
         <#anni>    40 ;
         <#coloreocchi> 'blu' .
```

sta a significare che Mario ha tre figli (Stefano, Luca e Paola), ha 40 anni e ha gli occhi blu.

2.1.2 OWL

Il Semantic Web è una visione del futuro del Web in cui all'informazione è dato un esplicito significato rendendo quindi più semplice alle macchine processare e integrare l'informazione disponibile sul Web in modo automatico. Il Semantic Web sarà, come già detto, costruito sulla capacità di XML di definire schemi di marcatura personalizzati e sulla flessibilità di RDF nella rappresentazione dei dati. Il primo livello al di sopra di RDF richiesto dal Semantic Web è un linguaggio per ontologie che possa definire formalmente il significato della terminologia utilizzata nei documenti Web. Se si vuole che le macchine possano eseguire dei ragionamenti sui documenti il linguaggio deve andare oltre la semplice semantica di RDF Schema.

OWL è stato progettato per soddisfare la necessità di un Web Ontology Language (linguaggio per ontologie per la rete). OWL è parte delle raccomandazioni del W3C riguardo al Semantic Web [16] [15] [7].

Che cos'è un'ontologia

Un'ontologia definisce i termini usati per descrivere e rappresentare una porzione di conoscenza. Le ontologie sono utilizzate da persone, database e ap-

plicazioni che hanno il bisogno di condividere un dominio di informazione (ad esempio sulla medicina, sull'ingegneria, economia, etc.). Le ontologie includono definizioni processabili dai computer di concetti semplici di un particolare dominio e le relazioni tra di essi. Codificano la conoscenza di un dominio ed anche quella di domini adiacenti. In questo modo rendono la conoscenza riutilizzabile.

Il termine ontologia viene utilizzato per descrivere artifatti di diversa complessità. Questi spaziano dalle semplici tassonomie (come le directory di Yahoo), agli schemi di metadata (come Dublin Core) alle teorie logiche. Il Semantic Web necessita di ontologie con gradi di complessità piuttosto complessi. Devono infatti servire per specificare i seguenti tipi di concetti:

- Classi nei più svariati domini di interesse
- Le relazioni che possono esistere tra le classi
- Le proprietà (o attributi) che queste classi possono avere

Le ontologie sono solitamente espresse in un linguaggio basato sulla logica, così che possano esistere distinzioni accurate, dettagliate, consistenti e significative tra le varie classi, proprietà e relazioni. Alcuni strumenti ontologici possono effettuare ragionamenti automatici utilizzando le ontologie e quindi fornire servizi avanzati ad applicazioni intelligenti come software per ricerche semantiche/concettuali, supporto per decisioni, riconoscimento di linguaggi naturali e vocali, database intelligenti.

Le ontologie sono prominenti nell'emergente Semantic Web e si presentano come un modo per rappresentare la semantica di documenti e di permetterne l'utilizzo da parte di applicazioni web e agenti software intelligenti. Le ontologie possono essere molto utili alle comunità come un modo per strutturare e definire il significato dei termini dei metadata che si stanno attualmente collezzionando e standardizzando. Usando le ontologie le applicazioni di domani potranno essere "intelligenti" nel senso che potranno lavorare più accuratamente al livello concettuale degli umani.

Le ontologie sono fondamentali per applicazioni che vogliano cercare o inglobare informazione da diverse comunità. Anche se XML [24], DTD [17] e XML-Schema [6] sono sufficienti per scambiare dati tra entità che hanno aderito sulle definizioni a priori, mancano della semantica impedendo di fatto alle macchine di eseguire il loro lavoro quando si trovano di fronte nuovi vocabolari

XML. Lo stesso termine può essere utilizzato con differenti significati in differenti contesti e termini differenti possono venire utilizzati per oggetti che hanno lo stesso significato. RDF e RDF-Schema permettono un approcio al problema permettendo di definire semplici semantiche utilizzando degli identificatori. Con RDF-Schema si possono definire classi con varie super-classi o sotto-classi e si possono definire proprietà con relative sotto-proprietà, domini e intervalli. In questo senso RDF-Schema è un semplice linguaggio ontologico. Comunque, per poter ottenere interoperabilità tra numerosi schemi sviluppati autonomamente, è richiesta una semantica più ricca. Per esempio RDF-Schema non può specificare che una Persona ed un'Auto sono classi distinte o che un quartetto d'archi ha esattamente quattro musicisti come membri.

I tre sottolinguaggi di OWL

OWL offre tre sottolinguaggi via via sempre più espressivi [7]:

OWL Lite utile agli utenti le cui principali necessità sono la classificazione gerarchica e la definizione di semplici vincoli. Per esempio supporta vincoli sulla cardinalità ma permette semplicemente valori di 0 o 1. Dovrebbe essere più semplice fornire strumenti che supportano OWL Lite piuttosto che per gli altri due sottolinguaggi e OWL Lite risulta essere un semplice e veloce metodo per far migrare vocabolari e tassonomie esistenti.

OWL DL (Description Logic) utile per coloro che vogliono la massima espressività restando comunque nei limiti della completezza computazionale (tutte le computazioni hanno luogo in un tempo finito), OWL DL include tutti i costrutti OWL ma possono essere utilizzati solo sotto certe condizioni (per esempio una classe può essere sottoclasse di molte classi ma una classe non può essere istanza di un'altra classe).

OWL Full è pensato per chi vuole il massimo dell'espressività e totale libertà sintattica ma senza garanzie computazionali. Per esempio in OWL Full una classe può essere trattata simultaneamente come una collezione di individui e come un singolo individuo. In ogni caso, sembra improbabile che qualche strumento software potrà mai supportare completamente tutte le caratteristiche di OWL Full.

Ognuno di questi sottolinguaggi è una estensione del linguaggio predecessore sia per quanto riguarda cosa sia possibile esprimere e sia per quanto riguarda

la complessità computazionale. Il seguente insieme di relazioni è valido, ma non il loro contrario:

- Ogni ontologia OWL Lite valida è una ontologia OWL DL valida.
- Ogni ontologia OWL DL valida è una ontologia OWL Full valida.
- Ogni conclusione OWL Lite corretta è una conclusione OWL DL corretta.
- Ogni conclusione OWL DL corretta è una conclusione OWL Full corretta.

Gli sviluppatori di ontologie che desiderano adottare OWL devono quindi scegliere quale sottolinguaggio adottare in base alle loro necessità (espressive e di computabilità). Nel modello descritto in questa tesi si è utilizzato OWL Lite.

2.1.3 SPARQL

RDF è un metodo flessibile ed estendibile per rappresentare informazione riguardo al World Wide Web. È utilizzato, tra l'altro, per rappresentare informazioni personali, reti sociali, e metadata riguardo a documenti digitali. Un linguaggio di interrogazione su RDF, standardizzato e con multiple implementazioni permetterebbe a sviluppatori ed utenti un modo potente di scrivere query su questa vasta mole di informazioni. SPARQL (definizione ricorsiva, SPARQL Protocol And RDF Query Language) [9] è sostanzialmente un linguaggio per effettuare query (interrogazioni) su basi di dati definite tramite RDF. Si tratta di un *working draft* del W3C in via di standardizzazione da parte del RDF Data Access Working Group [35].

Un grafo RDF è un insieme di triple; ogni tripla consta di un soggetto, un predicato ed un oggetto. Queste triple possono arrivare da varie sorgenti. Per esempio possono venire da un documento RDF, o possono essere inferite da altre triple o possono essere l'espressione RDF di dati presenti in altri formati come XML o un database relazionale.

SPARQL è un linguaggio d'interrogazione atto ad ottenere informazione da questi grafi RDF. Fornisce modalità per:

- estrarre informazione in forma di URI, blank nodes, e literals;

- estrarre sottografi RDF;
- costruire nuovi grafi RDF basandosi su informazioni presenti nel grafo interrogato.

Semplici query

Il linguaggio SPARQL è basato sull'idea di *matching graph patterns*. Il più semplice graph pattern è il pattern tripla che è come una tripla RDF solo che ha la possibilità di avere una variabile in una delle sue parti (soggetto, predicato, oggetto).

L'esempio qui sotto mostra una query SPARQL per trovare il titolo di un libro a partire dalle informazioni presenti in un grafo RDF. La query consiste di due parti. La clausola `SELECT` identifica le variabili che appariranno nel risultato e la clausola `WHERE` ha un pattern tripla.
Dati:

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title>
''SPARQL Tutorial''
```

Query:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
  <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Risultato:

title
''Sparql Tutorial''

Includere valori opzionali

I semplici graph pattern permettono alle applicazioni di compiere delle query in cui l'intero pattern deve essere soddisfatto per poter ottenere una soluzione. Per ogni soluzione della query ogni variabile è collegata ad un termine della

soluzione.
RDF è semistrutturato in quanto una struttura completa e regolare non può essere supposta per cui sarebbe utile poter trovare soluzioni ad una query anche quando non è completamente soddisfatta.

Le parti opzionali di un graph pattern possono essere specificate utilizzando la parola chiave `OPTIONAL`. Vediamo un esempio utlizzando i seguenti dati:

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:       <http://www.w3.org/2000/01/rdf-schema#> .

_:a  rdf:type        foaf:Person .
_:a  foaf:name       Alice .
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  rdf:type        foaf:Person .
_:b  foaf:name       Bob .
```

Si voglia creare una query per ricavare la proprietà `foaf:name` da qualsiasi risorsa dotata di tale proprietà e, nel caso sia presente, di venire a conoscenza anche della relativa proprietà `foaf:mbox`:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox }
       }
```

Il risultato sarà:

name	mbox
Alice	<mailto:alice@example.com>
Bob	

Quella descritta è una caratteristica molto utile per effettuare query sul modello descritto in questa tesi, in quanto molte proprietà delle classi descritte sono opzionali.
Purtroppo la libreria utilizzata per la creazione dei prototipi (vedi la prossima sezione) non risulta ancora completamente affidabile da questo punto di vista (vedi sez. 4.1.11).

2.1.4 Redland: librdf

Redland (o librdf) [2] è un RDF Application Framework, ossia un insieme di librerie e applicazioni nate per il trattamento di metadati RDF. Tra le sue caratteristiche le più importanti al fine di questo progetto sono:

- Fornisce una API (Application Program Interface) per il trattamento di grafi RDF con supporto per URI e Literal;
- Include un parser per la lettura di file RDF scritti in diversi formati (RDF/XML, N-Triples, Turtle);
- Permette di eseguire delle query utilizzando i linguaggi SPARQL e RDQL;
- È scritta in C ma fornisce dei binding con altri linguaggi quali C#, Java, Obj-C, Python, Perl e Tcl.

L'utilizzo di questa libreria ha quindi permesso di poter utilizzare Python come linguaggio di programmazione fornendo la possibilità di "leggere" i dati RDF da file esterni e di effettuare le query necessarie allo svolgimento dei compiti prefissati (anche se con qualche problema vedi sez. 4.1.11).

2.2 Python

Per lo sviluppo dei prototipi è stato utilizzato il linguaggio di programmazione Python [32] [14]. I motivi della sua scelta sono sostanzialmente due:

1. Facilità e velocità di programmazione. Sono due caratteristiche primarie da ricercare in un linguaggio adatto alla programmazione di prototipi.
2. Portabilità. Essendo il modello basato su tecnologie aperte e standard era importante continuare nella stessa direzione e non chiudersi in una singola piattaforma. Python è un linguaggio interpretato in grado di "girare" sulla maggior parte dei sistemi attualmente in circolazione. Stessa enfasi è stata data alla ricerca di librerie accessorie che rendessero quindi possibile la portabilità dei prototipi (rendendoli quindi "*cross-platform*").

Era stato preso in considerazione anche il linguaggio Java, che avrebbe garantito la portabilità ma sarebbe stato sicuramente molto più lungo il tempo necessario alla scrittura del codice e al suo debugging.

2.2.1 Piccola introduzione

Python è un potente linguaggio di programmazione interpretato creato da Guido van Rossum.

Si tratta di un linguaggio multi-paradigma. Infatti permette in modo agevole di scrivere programmi seguendo il paradigma object oriented, oppure la programmazione strutturata, oppure la programmazione funzionale. Il controllo dei tipi viene fatto a runtime (dynamic typing) e usa un garbage collector per la gestione automatica della memoria. Python ha qualche similarità con Perl, ma i suoi progettisti hanno scelto la via di una sintassi molto più essenziale e uniforme, con l'obiettivo di aumentare la leggibilità del codice. Come il Perl spesso è classificato linguaggio di scripting, ma pur essendo utile per scrivere script di sistema (in alternativa ad esempio a bash), la grande quantità di librerie disponibili e la facilità con cui questo linguaggio permette di scrivere software modulare favoriscono anche lo sviluppo di applicazioni molto complesse.

Python ha un gran numero di tipi base. Oltre ai tipi interi e floating point classici, supporta trasparentemente numeri interi arbitrariamente grandi e numeri complessi. Dalla versione 2.4 sono disponibili anche i numeri decimali (decimal), ovvero numeri con la virgola a precisione illimitata, come quelli disponibili in Rexx o in Cobol, che non soffrono di problemi di arrotondamento e stabilità tipici dei numeri floating point classici.
Supporta tutte le operazione classiche sulle stringhe con questa eccezione: le stringhe in Python sono oggetti immutabili, cosicché qualsiasi operazione che in qualche modo potrebbe alterare una stringa (come ad esempio la sostituzione di un carattere) restituirà invece una nuova stringa.
Essendo il Python a tipizzazione dinamica, tutte le variabili sono in realtà semplici puntatori ad oggetto (reference), sono gli oggetti invece ad essere dotati di tipo. Ad esempio ad una variabile cui era assegnato un intero, un istante dopo può essere assegnata una stringa o un array.
In Python c'è un moderato controllo dei tipi a runtime. Si ha conversione implicita per i tipi numerici, per cui si può ad esempio moltiplicare un numero complesso per un intero, ma non c'è ad esempio conversione implicita tra numeri e stringhe, per cui un numero è un argomento non valido per le operazioni su stringa.

Python ha una serie di tipi contenitori come ad esempio liste, tuple e dizionari. Liste, tuple e stringhe sono sequenze e condividono la maggior parte dei metodi: si può iterare sui caratteri di una stringa con la stessa facilità con

cui lo si può fare sugli elementi di una lista. Le liste sono array estendibili, invece le tuple sono array immutabili di lunghezza prefissata.
Altri contenitori di grande utilità sono i dizionari, conosciuti in altri contesti con il nome di hash table oppure array associativi. Come chiavi dei dizionari possono essere usati solo oggetti immutabili, in modo che in ogni caso sia preservata la consistenza, invece come valori associati alla chiave vanno bene oggetti arbitrari.

Una cosa inusuale del Python è il metodo che usa per delimitare i blocchi di programma, che lo rende unico fra tutti i linguaggi più diffusi.

Nei linguaggi derivati dall'Algol – come ad esempio Pascal, C e Perl – i blocchi di codice sono indicati con le parentesi oppure con parole chiave. (Il C ed il Perl usano ; il Pascal usa begin ed end). In questi linguaggi è solo una convenzione degli sviluppatori il fatto di indentare il codice interno ad un blocco, per metterlo in evidenza rispetto al codice circostante.
Python, invece, prende a prestito una caratteristica dal meno noto linguaggio di programmazione Occam – invece di usare parentesi o parole chiavi, usa l'indentazione stessa per indicare i blocchi nidificati. Di seguito un esempio per chiarire questo. La versione C e Python di funzioni che fanno la stessa cosa – calcolare il fattoriale di un intero:
Fattoriale in C:

```
int fattoriale(int x) {
    if (x == 0) {
        return 1;
    } else {
        return x * fattoriale(x-1);
    }
}
```

Fattoriale in Python:

```
def fattoriale(x):
    if x == 0:
        return 1
    else:
        return x * fattoriale(x-1)
```

2.3 wxWidgets

Per fornire un prodotto cross-platform è necessario sviluppare l'interfaccia utente utilizzando una libreria per la costruzione di GUI (Graphical User Interface) che sia il più possibile portabile e che rispetti nel miglior modo possibile il look-and-feel (ossia l'aspetto) del sistema operativo su cui viene fatta girare. In questo wxWidgets [39] è una delle più consolidate e utilizzate librerie open-source.

I suoi punti di forza utili nella realizzazione del progetto sono:

- Piuttosto semplice da utilizzare, aiutata in questo da un'ottima ed estensiva documentazione.
- Nata più di dodici anni fa e in costante sviluppo. Questo ne garantisce l'affidabilità e il supporto.
- Sta diventando uno standard nello sviluppo di interfacce utente con Python.
- Più che buono il rispetto del look-and-feel del sistema operativo ospitante.

Capitolo 3

Il modello

Di seguito verrà presentato il modello concettuale che sta alla base del trattato. Si parlerà del ruolo fondamentale che hanno i metadata nel descrivere l'ambiente operativo, della rappresentazione dei mime-tpye dei documenti utilizzando RDF e OWL, dell'aggiunta di metadata alle applicazioni e del motore inferenziale necessario al progetto.

3.1 Metadati

In questo progetto i metadati hanno un ruolo fondamentale. Si può dire che ne sono le fondamenta. La definizione formale di "metadati" è "dati riguardanti dati", ad esempio il catalogo di una libreria è una raccolta di metadati in quanto contiene informazioni (dati) riguardanti delle altre pubblicazioni (altri dati). I metadati serviranno quindi a descrivere l'ambiente operativo di un normale computer desktop. In particolare conterranno informazioni riguardanti i documenti degli utenti e sulle applicazioni installate nel sistema.

Tali informazioni verranno rappresentate utilizzando un recente linguaggio di definizione e organizzazione di metadata proposto e standardizzato dal W3C [38] (vedi capitolo 2).

3.2 Mimetype

Il primo passo da affrontare per definire il modello è quello di categorizzare i possibili documenti in base al loro mime-type [29]. Molti sistemi (o am-

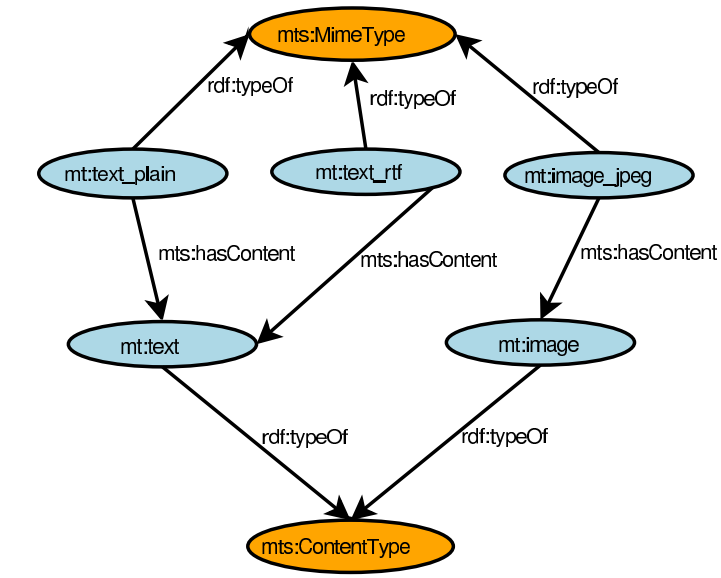

Figura 3.1: Mime-type e Content-type

bienti) operativi mettono a disposizione un modo per capire il mime-type dei documenti tramite alcune caratteristiche come l'estensione del nome del file (Windows) o anche dall'intestazione del loro contenuto (Gnome).

Una volta scoperto il mime-type di un documento è possibile rappresentarlo attraverso i costrutti messi a disposizione dai linguaggi RDF [10] e OWL [16]. In particolare è stato creato un vocabolario in cui sono definite alcune classi OWL con relative proprietà così da poter associare una URI ad ogni possibile mime-type e quindi virtualmente ad ogni documento prodotto e/o gestibile dall'utente (vedi fig. 3.1).

3.2.1 Lo schema

La classe MimeType contiene informazioni riguardo al nome del mime-type che rappresenta, alla relativa classe rappresentante il content-type (che non ha

alcun attributo, per cui non ne viene riportata la tabella) e alle operazioni che possono essere eseguite su di essa (vedi sezione Operazioni (3.4)).

Classe OWL `MimeType`	
Proprietà	**Commento**
`name`	Nome del mime-type (es. "text/plain" o "image/jpeg").
`hasContent`	URI del relativo content-type.
`canApply`	URI ad una operazione.

Tabella 3.1: Classe OWL `Mimetype`

Di seguito viene riportato il codice RDF-XML delle classi sopra descritte al fine di illustrare la modalità di scrittura di un semplice vocabolario OWL:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xml:base="http://sflow.org/schema/2005-06/mimetypes">

<owl:Class rdf:ID="ContentType">
  <rdfs:label xml:lang="it">Content Type</rdfs:label>
  <rdfs:label xml:lang="en">Content Type</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="MimeType">
  <rdfs:label xml:lang="it">Mime Type</rdfs:label>
  <rdfs:label xml:lang="en">Mime Type</rdfs:label>
  <rdfs:subClassOf rdf:resource="#ContentType" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#MimeType" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasContent">
  <rdfs:domain rdf:resource="#MimeType" />
  <rdfs:range rdf:resource="#ContentType" />
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="canApply">
  <rdfs:domain rdf:resource="#MimeType" />
  <rdfs:range rdf:resource="http://sflow.org/schema/2005-06/operations↩
      #Operation" />
</owl:ObjectProperty>

</rdf:RDF>
```

Esempio

A titolo di esempio viene illustrato come definire dei content-type e dei mime-type utilizzando il vocabolario precedentemente descritto. In particolare con il seguente codice vengono create le classi (e quindi le URI) che vanno a rappresentare il content-type "text" e il mime-type "text/plain".

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:mm="http://sflow.org/schema/2005-06/mimetype#"
  xml:base="http://sflow.org/data/2005-06/mimetype">

<mm:ContentType rdf:ID="text" />

<mm:MimeType rdf:ID="text_plain">
  <mm:name>text/plain</mm:name>
  <mm:hasContent rdf:resource="#text" />
</mm:MimeType>

</rdf:RDF>
```

3.3 Applicazioni

Avendo ottenuto informazioni sui file del sistema possiamo estendere il modello aggiungendo metadati sulle applicazioni. È stato quindi definito un altro vocabolario che tramite delle opportune classi OWL con relative proprietà permetterà di arricchire di informazioni la base semantica precedentemente creata. È in effetti semplice immaginare di legare le applicazioni presenti nel sistema con i mime-type che sono in grado di gestire, specificando se in sola lettura o anche in scrittura (vedi fig. 3.2).

In realtà si può andare oltre ed aggiungere ulteriori informazioni; ogni applicazione potrebbe infatti specificare dati riguardanti la propria installazione

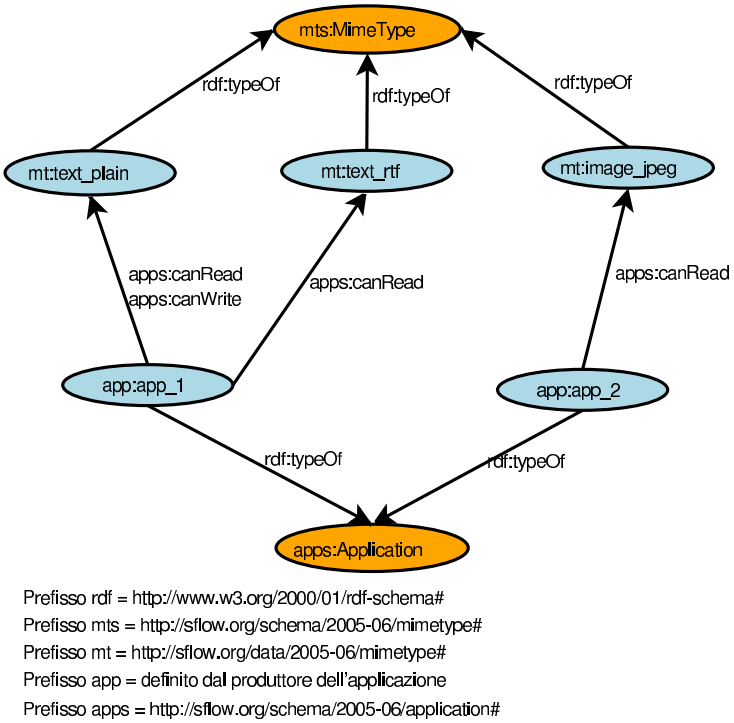

Figura 3.2: Mime-type e Applicazioni

(path di installazione, nome, versione), il proprio tipo di licenza (open source, freeware, commerciale) e informazioni accessorie (pagina web, prezzo). Tutto ciò servirà ad interrogare il sistema con opportune query così da rendere più ricca e semplice l'interazione con l'utente.

Già a questo punto si potrebbe cominciare a porre delle domande alla base semantica creata. La più ovvia è quella di "domandare" quali applicazioni siano in grado di leggere o modificare un dato documento. Alcuni sistemi operativi mettono già a disposizione un sistema simile ma non tutti ed ognuno lo implementa differentemente utilizzando tecnologie più o meno proprietarie. Con la tecnica sopra citata si risolverebbe il problema in maniera semplice, pulita e soprattutto standard.

3.3.1 Lo schema

Di seguito è riportata, in forma tabellare, la prima parte del vocabolario relativo alle applicazioni; la seconda parte verrà trattata assieme alle Operazioni nella sezione successiva (3.4).

Tabella 3.2: Classe OWL `Application` (prima parte)

Classe OWL `Application` parte 1	
Proprietà	**Commento**
`name`	Nome dell'applicazione.
`license`	Tipo di licenza (GPL, freeware, commerciale, etc.). Si tratta di una proprietà opzionale.
`price`	Prezzo dell'applicazione (proprietà opzionale).
`webURL`	Indirizzo web dell'applicativo (proprietà opzionale).
`version`	Versione del programma (proprietà opzionale).
`hasGUI`	Valore booleano, vero se l'applicativo ha una interfaccia utente, falso altrimenti (proprietà opzionale).
`installationPath`	Path in cui si trova l'eseguibile dell'applicazione.
`execCommand`	Nome dell'eseguibile dell'applicativo.
`canRead`	URI dei mime-type in grado di essere letti dall'applicazione.
`canWrite`	URI dei mime-type in grado di essere modificati dall'applicazione.

Esempio

Il seguente è il codice necessario alla definizione dell'applicativo "Evince". Si tratta di un software open-source per il sistema operativo GNU/Linux in grado

di leggere documenti PDF, PostScript e DVI.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:mm="http://sflow.org/data/2005-06/mimetype#"
  xmlns:apps="http://sflow.org/schema/2005-06/applications#"
  xml:base="http://sflow.org/data/2005-06/application">

<apps:Application rdf:ID="evince">
  <apps:name>Evince</apps:name>
  <apps:installationPath>/usr/bin/</apps:installationPath>
  <apps:execCommand>evince</apps:execCommand>
  <apps:version>0.3</apps:version>
  <apps:hasGUI>true</apps:hasGUI>
  <apps:license>Open-Source</apps:license>
  <apps:webURL>http://www.gnome.org/projects/evince/</apps:webURL>
  <apps:canRead rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      application_pdf"/>
  <apps:canRead rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      application_postscript"/>
  <apps:canRead rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      application_x-dvi"/>
</apps:Application>

</rdf:RDF>
```

3.4 Operazioni

Il contributo maggiore, sia in termini di quantità che di utilità delle informazioni che possiamo raccogliere nella base semantica che stiamo creando viene dalla descrizione delle operazioni. Ogni applicazione potrebbe infatti rendere note, sempre tramite costrutti RDF/OWL, le singole operazioni che è in grado di compiere specificandone i dettagli. Ad esempio un programma di fotoritocco potrebbe rendere nota la propria capacità di applicare un filtro bianco e nero a particolari tipi di immagini (vedi fig. 3.3).

In questo modo le query applicabili alla base semantica diventano molto più potenti ed interessanti. Non diventa solo possibile chiedere quali applicazioni trattano un particolare tipo di documento ma anche quali operazioni sono in grado di compiere sul singolo mime-type. Così facendo l'applicazione passa in secondo piano lasciando protagoniste le singole operazioni. Partendo da un documento potremmo sapere quali operazioni possiamo compiere su di esso

senza interessarci di quali applicazioni installate nel nostro sistema siano in grado di compierle. È un primo passo verso il desktop "operation-oriented".

3.4.1 Parametri

È da tenere presente che non tutte le operazioni sono semplici come la conversione in bianco e nero di un'immagine. Molte richiederanno dei parametri in ingresso ed è quindi opportuno prevedere delle proprietà per rappresentare i parametri in dettaglio (vedi fig. 3.4) dando la possibilità di indicare il numero, il tipo e l'ordine dei parametri necessari allo svolgimento delle operazioni.

Il fatto di maggior rilievo quando si parla di parametri in ingresso è la loro identificazione, bisogna cioè fare in modo che l'applicazione capisca quali parametri le sono passati in input. In realtà non si tratta di un grosso problema in quanto esistono da sempre applicazioni in grado di trattare operazioni e relativi parametri dalla linea di comando, basta solo fare un po' di ordine in quanto generalmente ogni applicativo utilizza un'ordine e una sintassi diversa. È quindi necessario porre alcune semplici regole in modo da poter parlare un linguaggio comune, in questo caso ci si è ispirati alla *GNU Command Line Syntax*:

- I file in input andranno presentati subito dopo il comando dell'applicazione;
- I parametri possono venire identificati in due modi, o `-x` con `x` lettera singola oppure con `--xyz` con `xyz` parola qualunque. Possono essere seguiti da valori numerici o stringhe;
- Gli eventuali documenti in output vanno segnalati con `--output <nomi dei file>` o `-o <nomi dei file>`.

Si veda il capitolo Prototipi nella sezione 4.1.6 per alcuni esempi.

3.4.2 Categorie

Ai giorni nostri un normale computer desktop è solitamente fornito con un nutrito numero di applicazioni a corredo. Molte applicazioni implicano molte operazioni possibili sui vari tipi di documento. Data la lunga lista e avendo la necessità di doverle presentare all'utente al fine di permettergli di effettuare

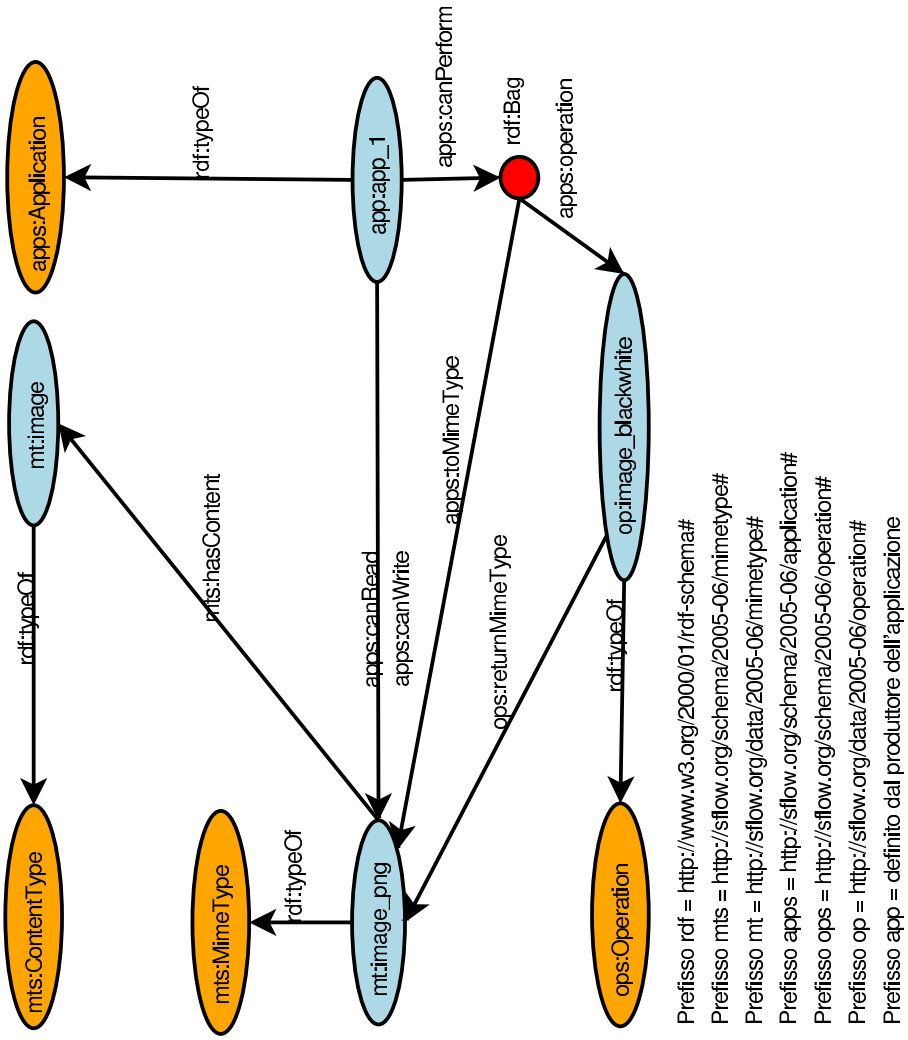

Figura 3.3: Applicazioni e operazioni

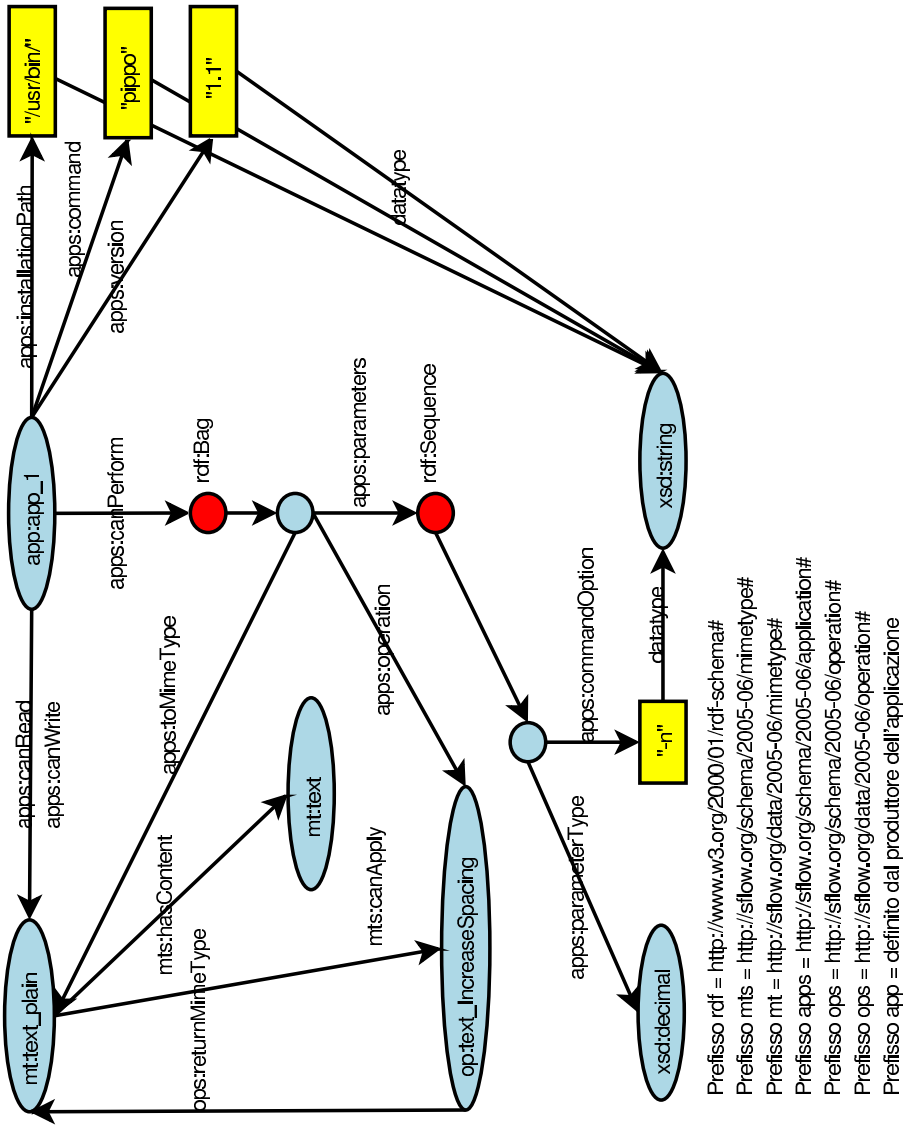

Figura 3.4: Operazioni in dettaglio

una scelta si correrebbe il rischio di sovraccaricarlo di informazioni. Al fine di evitare confusione si è fatto in modo che le operazioni possano venire categorizzate in modo gerarchico.

Sostanzialmente è stata creata una classe OWL rappresentante una generica categoria. Tale classe è dotata della proprietà `hasFather` che "punta" ad eventuali sopra-categorie così da poter creare una struttura gerarchica che sarà di più facile consultazione per l'utente.

3.4.3 Livello di complessità

Due software che sanno trattare uno stesso mime-type si possono differenziare in base al loro livello di complessità. Spesso infatti bastano applicazioni "leggere" per portare a compimento le operazioni che abbiamo in mente, lasciando ai software ben più pesanti il compito di effettuare operazioni più complesse.

Per questo motivo è stata definita la proprietà RDF (opzionale) `complexityLevel` che permette di specificare il livello di complessità dell'applicazione in una scala da 1 a 5.

3.4.4 Gli schemi

Viste le considerazioni precedenti andranno definite nuove classi e proprietà che permettano di descrivere le operazioni, le categorie e i parametri.

Tabella 3.3: Classe OWL `Category`

Classe OWL `Category`	
Proprietà	**Commento**
`name`	Nome della categoria.
`hasFather`	URI delle eventuali categorie padre.

Tabella 3.4: Classe OWL `Operation`

Classe OWL `Operation`	
Proprietà	**Commento**
`name`	Nome dell'operazione.
`returnMimeType`	URI del mime-type risultato dell'operazione. Ad esempio l'operazione di conversione di un file audio wave in un file audio mp3 avrà come mime-type di ritorno un documento audio/mpeg.
`groupOperation`	Valore booleano vero se l'operazione richiede uno o più di un file in input e ritorna un solo file in output, false altrimenti.
`shortComment`	Breve frase illustrativa dell'operazione (proprietà opzionale).
`hasCategory`	URI ad un'eventuale categoria di appartenenza (proprietà opzionale).

Inoltre va arricchita la classe Application descritta nella sezione precedente. Vengono quindi definite le nuove proprietà:

Tabella 3.5: Classe OWL `Application` (seconda parte)

Classe OWL `Application` parte 2	
Proprietà	**Commento**
`complexityLevel`	Livello di complessità dell'applicazione espresso con un valore da 1 a 5 (proprietà opzionale).
`canPerform`	segue una collezione non ordinata di blank node aventi proprietà `toMimeType` e `operation`.
`operation`	URI di un'operazione.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`toMimeType`	URI del/i mime-type su cui è in grado di compiere l'operazione.
`URLComment`	URL ad una pagina (X)HTML in cui viene fornito un help approfondito per l'operazione (parametro opzionale).
`parameters`	una sequenza ordinata di blank-node descriventi i parametri necessari all'operazione. A tal fine vengono utilizzate le proprietà successive.
`commandOption`	Stringa identificativa del parametro (ad esempio `--convert`).
`parameterType`	Tipo del parametro (stringa, intero, boolean), la proprietà è opzionale in quanto non tutti i parametri richiedono di specificare dei dati. In ogni caso si deve far riferimento ai tipi di dato definiti in XML-Schema [6].
`parameterOrder`	Specifica un ordinamento tra i parametri (vedi sez. 4.1.11). Si tratta di un parametro opzionale.
`parameterLabel`	Se il parametro richiede dei dati in input (ad esempio per il ridimensionamento di un'immagine bisogna conoscere il nuovo valore per la dimensione) questa propretà rappresenta una stringa che verrà proposta all'utente in modo da chiarire a cosa serve il parametro (ad esempio "Nuova dimensione dell'immagine:"). Si tratta di una proprietà opzionale.
`parameterHelp`	Piccola stringa di aiuto relativa al parametro (parametro opzionale).

Esempio

Volendo descrivere l'operazione di ridimensionamento di un'immagine tramite l'utlizzo del pacchetto di programmi a linea di comando ImageMagick [30] si può procedere scrivendo un documento RDF/XML utilizzando i vocabolari sopra descritti. Si otterrebbe quindi una risorsa come la seguente:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:mm="http://sflow.org/data/2005-06/mimetype#"
  xmlns:apps="http://sflow.org/schema/2005-06/applications#"
  xmlns:ops="http://sflow.org/schema/2005-06/operations#"
  xml:base="http://www.imagemagick.org/applications">


<ops:Operation rdf:ID="imageResize">
  <ops:name>Ridimensiona</ops:name>
  <ops:returnMimeType rdf:resource="http://sflow.org/data/2005-06/↩
      mimetype#sameasinput" />
  <ops:shortComment>Ridimensionamento di un immagine</ops:shortComment↩
      >
</ops:Operation>


<apps:Application rdf:ID="convert">
  <apps:name>Convert</apps:name>
  <apps:installationPath>/usr/bin/</apps:installationPath>
  <apps:execCommand>convert</apps:execCommand>
  <apps:version>6.0</apps:version>
  <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      image_jpeg" />
  <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      image_png" />
  <apps:canPerform>
    <rdf:Bag>
      <rdf:li rdf:nodeID="_convertImageResize" />
      </rdf:Bag>
  </apps:canPerform>
</apps:Application>

<rdf:Description rdf:nodeID="_convertImageResize">
  <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
      mimetype#image_jpeg" />
  <apps:operation rdf:resource="#imageResize" />
  <apps:parameters>
    <rdf:Seq>
      <rdf:li rdf:nodeID="_convertImageResize_p1" />
    </rdf:Seq>
  </apps:parameters>
```

```
</rdf:Description>

<rdf:Description rdf:nodeID="_convertImageResize_p1">
  <apps:commandOption>--resize</apps:commandOption>
  <apps:parameterLabel>Nuova dimensione della larghezza</↩
      apps:parameterLabel>
  <apps:parameterType rdf:resource="&xsd;decimal" />
  <apps:parameterHelp>L'immagine àverr ridimensionata in base al nuovo
  valore della larghezza mantenedo le proporzioni</apps:parameterHelp>
</rdf:Description>
</rdf:RDF>
```

Particolarità

Da notare l'utilizzo del mime-type fittizzio `sameasinput` nella definizione dell'operazione imageResize. In questo modo si indica che il mime-type risultato dell'operazione è lo stesso di quello in ingresso. In realtà nulla ci vieterebbe di definire tutti i possibili mime-type in uscita per ogni operazione ma si andrebbe ad appesantire inutilmente la base semantica con l'unico risultato di rendere molto lenti gli applicativi che ne fanno uso.

Oltre al sopracitato `sameasinput` risulta molto utile un altro mime-type creato ad-hoc, ossia `every`. Si pensi infatti ad un programma di compressione di file (come ad esempio Gzip); questo applicativo può svolgere le proprie funzionalità su qualsiasi tipo di documento quindi dovrebbe essere specificata la proprietà `canRead` per ogni tipo di mime-type, presente e futuro. Oltre ad essere oneroso sarebbe inoltre computazionalmente pesante. Ecco quindi la necessità di definire un mime-type convenzionale che andasse a rappresentare qualsiasi tipo di mime-type, a tale mime-type è stato, appunto, dato il nome di `every`.

3.4.5 Il problema delle operazioni copia

Il modello fin qui descritto fornisce informazioni utili ma c'è un problema di fondo da risolvere. Qualora ogni applicazione presentasse le proprie operazioni in maniera indipendente, avremmo per ogni operazione una URI differente, anche per quelle che sono concettualmente uguali. Ad esempio il programma di fotoritocco $F1$ potrebbe identificare l'operazione di conversione in bianco e nero con l'URI `http://f1.com/operations#bw` mentre l'applicazione $F2$ con `http://www.f2.com/prodotti/operazioni#bn` sebbene in realtà l'operazio-

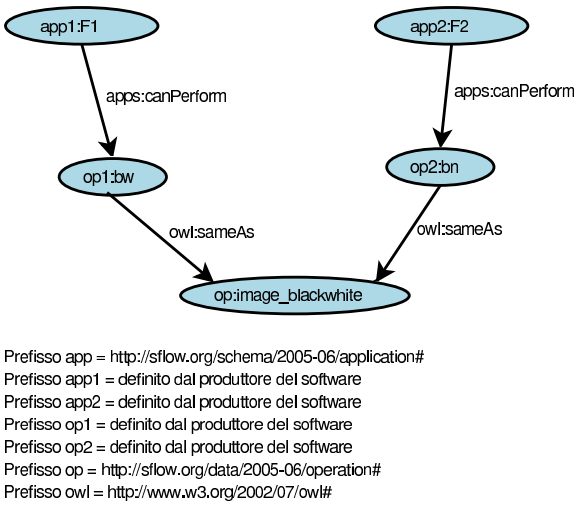

Figura 3.5: Equivalenze tra operazioni

ne sia la stessa. Per risolvere il problema viene utilizzato OWL (vedi sez. 2.1.2) che permette la definizione di uguaglianze tra oggetti semantici.

Infatti, se oltre a permettere alle applicazioni di presentare le proprie operazioni venisse definito anche un insieme di operazioni "standard" super-partes ecco che le operazioni più comuni, presenti in molti software diversi, potrebbero venir equiparate alla relativa operazione standard. Riprendendo l'esempio del filtro bianco e nero visto prima si può definire l'operazione standard di conversione in bianco e nero avente URI:
`http://sflow.org/data/2005-06/operation#image_blackwhite`
e equipararla alle URI viste precedentemente (vedi fig. 3.5). L'equivalenza è resa possibile grazie al linguaggio OWL ed in particolare utilizzando la proprietà `sameAs` che crea un legame di uguaglianza tra due classi distinte.

3.5 Il modello finale

Dato che ogni operazione segnala il tipo di mime-type in output è possibile creare un "*flusso*" di operazioni a partire da uno o più documenti (vedi fig. 3.6).

I vantaggi del modello sono principalmente due:

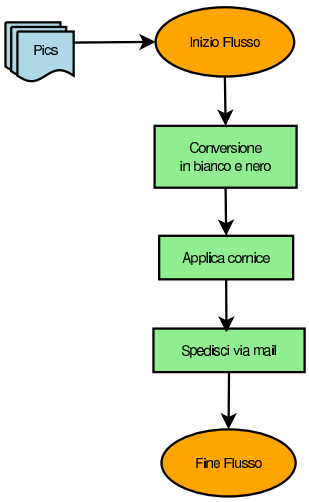

Figura 3.6: Esempio di flusso di operazioni

1. "Nascondere" il concetto di applicazione rendendo più evidenti le funzionalità delle operazioni. Non ci si dovrà più domandare "Con quale applicazione posso fare questa operazione?".

2. Dare un'interfaccia comune a qualsiasi tipo di operazione su qualsiasi tipo di documento. Non occorrerà più destreggiarsi tra diversi tipi di applicazioni per compiere una serie di operazioni su dei documenti.

3. Dare nuova enfasi alle applicazioni a linea di comando in quanto l'interfaccia utente verrà creata dinamicamente in base alle informazioni semantiche fornite.

3.5.1 Ulteriori vantaggi

Quando si ha a che fare con basi di dati (semantiche o meno) tutto quello che si può fare è frutto di opportune query su di esse. Molto spesso quando se ne costruisce una lo si fa avendo in mente il tipo di query che vi si andranno a fare ma, se la base è ben formata, ciò non vieta di potersene inventare delle altre successivamente. Nel modello sopra descritto si ha come interessante *effetto collaterale* la possibilità di creare una sorta di *help* dinamico delle applicazioni. Ogni applicazione infatti fornisce informazioni su se stessa (vedi sezione 3.3)

e sulle operazioni che è in grado di compiere (vedi sezione 3.4), risulta quindi semplice creare un'applicazione che sfruttando tali dati sia in grado di fornire informazioni su qualsiasi altro software installato nel sistema. Un prototipo di questo tipo di applicativo verrà illustrato in dettaglio nel capitolo 4 nella sezione 4.2.

Capitolo 4

I prototipi

In questo capitolo verranno illustrati i prototipi di due applicazioni che sfruttano la base semantica descritta precedentemente: Sflow e AppHelp.

4.1 Sflow

Sflow è il nome dato al prototipo dell'applicazione implementante l'idea del flusso di operazioni visto nel capitolo precedente (vedi fig. 3.6). Il nome deriva dalla fusione delle parole Semantic e Flow ossia "Flusso semantico".

4.1.1 Requisiti

Vengono di seguito illustrati i requisiti funzionali e non funzionali alla base del prototipo. Va in ogni caso precisato che sia il modello che i prototipi non sono stati sviluppati in team ma da un'unica persona rendendo quindi quasi impossibili alcune fasi importanti dell'ingegneria del software come la discussione con il committente, le fasi di brainstorming e di studio di gruppo, nonché l'impossibilità di creare delle figure distinte per l'analisi, la fase di testing e di gestione della documentazione. Ne segue quindi un'analisi dei requisiti più semplice di quella che sarebbe emersa lavorando con un gruppo di persone.

Requisiti funzionali

I requisiti funzionali descrivono i servizi forniti dal sistema, come reagisce agli input e come si comporta in determinate situazioni. Devono anche esplicitare quello che il sistema non deve fare.
Di seguito vengono riportati in forma tabellare i requisiti funzionali per l'applicativo Sflow:

Tabella 4.1: Requisisti funzionali

	Requisito	**Spiegazione**
RF1	Effettuare il parsing di documenti RDF	Al fine di svolgere le proprie funzionalità Sflow dovrà essere in grado di "leggere" dei documenti RDF inglobandone le triple in un'unica base semantica.
RF2	Query su dati RDF	Sflow dovrà poter effettuare delle interrogazioni sulla base semantica creata.
RF3	Ricevere uno o più file in input	L'applicativo dovrà poter ricevere uno o più documenti in input attraverso la linea di comando.
RF4	Presentare una lista di applicativi	Basandosi sulle informazioni presenti nella base di dati semantica Sflow dovrà presentare all'utente una lista delle applicazioni in grado di leggere o modificare tutti i documenti forniti in ingresso.
RF5	Eseguire applicativo esterno	Dalla lista illustrata nel punto precedente l'utente dovrà avere la possibilità di scegliere un applicativo e di aprire i documenti in ingresso con l'applicazione scelta.
Continua nella pagina successiva		

Continua dalla pagina precedente

RF6	Aggiungere operazioni	L'utente dovrà avere la possibilità di scegliere delle operazioni effettuabili sui documenti in input dalle applicazioni esterne in grado di manipolarli.
RF7	Flusso di operazioni	Le operazioni dovranno poter essere aggiunte una di seguito all'altra al fine di formare un flusso (o una *pipeline*) di operazioni.
RF8	Operazioni divise per categorie	È da prevedere la possibilità per le operazioni di appartenere a qualche forma di categoria.
RF9	Categorie gerarchiche	Le categorie devono avere la possibilità di essere strutturate in forma gerarchica.
RF10	Gestire i parametri delle operazioni	Qualora un'operazione abbia bisogno di parametri in input Sflow dovrà essere in grado di richiederli all'utente e passarli all'applicativo che li richiede.
RF11	Informazioni sulle operazioni	Qualora siano presenti nella base di dati delle informazioni di aiuto riguardanti le operazioni Sflow dovrà renderle disponibili all'utente.
RF12	Gestione nomi dei documenti in output	Qualora il flusso di operazioni produca dei documenti l'utente dovrà specificarne il nome.
RF13	Complessità delle applicazioni	Le applicazioni esterne ad Sflow devono avere la possibilità di essere categorizzate in base alla loro complessità e l'utente dovrà poter scegliere il livello di complessità desiderato.
RF14	Possibilità di salvare e ricaricare i flussi	L'utente dovrà poter salvare un flusso di lavoro (operazioni e valori di eventuali parametri) per poterlo poi richiamare in qualsiasi altro momento.

Continua nella pagina successiva

Continua dalla pagina precedente		
RF15	Cross-Platform	L'applicazione dovrà poter essere eseguibile senza particolari accorgimenti su piattaforme Windows, MacOS X e GNU/Linux.

Requisiti non funzionali

I requisiti non funzionali esplicitano i vincoli sui servizi forniti dal sistema e sulle modalità operative.
Di seguito vengono riportati in forma tabellare i requisiti non funzionali dell'applicativo Sflow:

Tabella 4.2: Requisisti non funzionali

	Requisito	**Spiegazione**
RNF1	Porre in evidenza gli applicativi in sola lettura da quelli anche in scrittura	La lista delle applicazioni in grado di gestire i documenti passati in ingresso ad Sflow dovrà presentare in modo diverso le applicazioni in sola lettura da quelle in grado anche di modificare i documenti in questione. *Questo è l'unico requisito non soddisfatto da Sflow. L'idea era quella di colorare in modo diverso le due categorie di applicativi ma sarebbe stato necessario un lungo lavoro di modifica alla libreria uxWidgets.*
RNF2	Gestione mime-type ignoti	Qualora Slow non riesca a stabilire il mime-type di qualche documento ricevuto in ingresso dovrà chiederlo all'utente.
RNF3	Possibilità di cambiare operazione	L'utente dovrà poter cambiare in qualsiasi momento le operazioni scelte.
Continua nella pagina successiva		

Continua dalla pagina precedente		
RNF4	Possibilità di rimuovere operazione	L'utente dovrà avere la possibilità di rimuovere delle operazioni dal flusso.
RNF5	Verifiche al cambio di operazione	Ogni qualvolta ci sia un cambio di operazione Sflow dovrà verificare che i mime-type degli eventuali documenti risultanti siano compatibili con le operazioni successive nel flusso.
RNF6	Possibilità di cambiare la fonte dei documenti RDF	L'utente dovrà avere la possibilità di scegliere, e cambiare successivamente, la directory in cui sono memorizzati i documenti RDF.
RNF7	L'interfaccia utente deve essere semplice	Grande importanza va data alla semplicità e linearità dell'interfaccia utente di Sflow. Linea guida principale deve essere la maggiore autonomia possibile di Sflow nel compiere decisioni.
RNF8	L'interfaccia di Sflow non dovrà mai superare lo spazio visibile dello schermo dell'utente	Essendo Sflow un'applicazione pensata per creare dei *flussi* la sua interfaccia utente sarà dinamica e tenderà ad occupare via via sempre più area dello schermo dell'utente. È da evitare assolutamente lo sconfinamento oltre i bordi dell'area visibile.
RNF9	Pulsante per lanciare applicativo esterno	Dovrà esistere un pulsante per permettere all'utente di aprire i documenti con uno degli applicativi esterni in grado di gestirli.
RNF10	Pulsante per aggiungere operazioni	È necessaria la presenza di un pulsante per aggiungere nuove operazioni al flusso di lavoro.
RNF11	Pulsante per eseguire il flusso	Dovrà essere presente un pulsante avente il compito di eseguire il flusso di operazioni definito dall'utente.
Continua nella pagina successiva		

Continua dalla pagina precedente		
RNF12	Nascondere pulsanti	Qualora sia presente una o più operazioni nel flusso di lavoro, il pulsante relativo all'applicativo esterno e la lista degli applicativi esterni dovranno essere nascosti.
RNF13	Interfaccia diversa per la gestione dei nomi in output in base al numero di documenti in uscita	L'utente dovrà capire, in base a delle segnalazioni, quando i documenti in uscita dal flusso siano più di uno.
RNF14	Pulsante per ottenere maggiori informazioni	Qualora la base di dati fornisca informazioni aggiuntive riguardanti un'operazione queste dovranno essere accessibili attraverso un apposito pulsante.
RNF15	Visualizzazione andamento del flusso	Quando il flusso di operazioni verrà eseguito l'utente dovrà essere informato del suo andamento.

4.1.2 Funzionamento

Si parte fornendo in ingresso all'applicazione uno o più documenti. Partendo da questi il programma cerca di determinarne il mime-type e qualora non ci riuscisse verrebbe chiesto aiuto all'utente, vedi fig. 4.1. In base al mime-type ottenuto verrà presentata all'utente una lista di applicazioni in grado di aprire i documenti permettendo, inoltre, all'utente di aprirli con l'applicazione scelta; sostanzialmente viene fornita una funzione *Apri con* intelligente, non ancora presente in tutti i sistemi operativi attuali (vedi fig. 4.2).

Qualora l'utente non voglia modificare i documenti utilizzando le applicazioni proposte (o perché non le conosce bene o perché non sa ancora bene cosa voler fare e quindi non è in grado di scegliere l'applicazione giusta) è sempre presente il pulsante "Aggiungi operazione" che farà apparire un nuovo *pannello delle operazioni*. Questo pannello presenta una lista categorizzata delle possibili operazioni eseguibili dalle applicazioni installate nel sistema sui documenti dati in ingresso. Da notare che la lista contiene **solamente le operazioni**

Figura 4.1: Sflow: Finestra di dialogo per mime-type ignoti

Figura 4.2: Sflow: "Apri con" intelligente

Figura 4.3: Sflow: esempio di operazione

possibili per i documenti forniti in input e non una semplice lista di tutte le operazioni proposte dalle applicazioni installate. L'utente può navigare tra le categorie e scegliere l'operazione desiderata. Una volta effettuata la scelta, nel caso l'operazione abbia bisogno di uno o più parametri per essere portata a termine, questi verranno richiesti all'utente (vedi fig. 4.3).

L'operazione scelta potrebbe fornire uno o più documenti come risultato della sua esecuzione e questo permette di costruire il *flusso* di operazioni. Infatti premendo ancora il pulsante "Aggiungi Operazione" è possibile continuare ad aggiungere nuove operazioni in cascata (vedi fig. 4.4).

Completata la definizione, il flusso può essere eseguito premendo il pulsante "GO!". Ad ogni operazione intermedia verranno creati dei documenti temporanei che vengono passati in input all'operazione successiva. Tali documenti vengono quindi cancellati dal sistema una volta che il flusso di operazioni è concluso.
Nella sezione 4.1.8 si potranno trovare degli esempi di possibili flussi.

4.1.3 Le classi del modello

Di seguito vengono illustrate in forma tabellare le classi Python create così da poter sfruttare il modello descritto nel capitolo precedente.

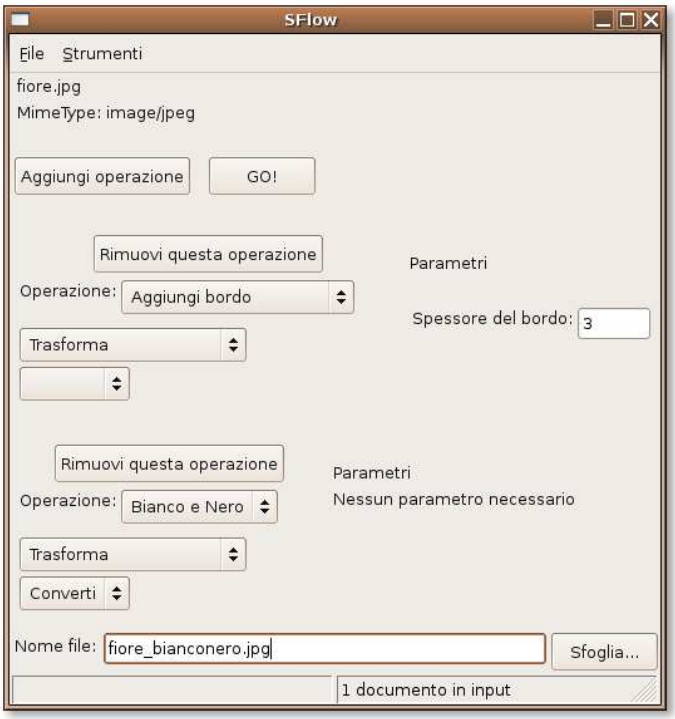

Figura 4.4: Sflow: Operazioni in cascata

Tabella 4.3: Classe Mimetype

Classe `Mimetype`	
Classe base per la gestione dei Mime-Type. Ha il compito di recuperare dalla base semantica le applicazioni in grado di lavorare sul Mime-Type in questione.	
Metodo	**Commento**
`init(mimetype, model, complexityLevel)`	Costruttore. Viene passata in ingresso la stringa rappresentante il mimetype, ad esempio 'text/plain' , un oggetto RDF.Model della libreria librdf rappresentante l'handler per la base semantica e un numero intero che indica il livello di complessità desiderato (vedi sez. 3.4.3).
`GetURIAppsCanRead()`	Ritorna una lista di URI di applicazioni in grado di leggere il mimetype.
`GetURIAppsCanReadAll()`	Ritorna una lista di URI di applicazioni in grado di leggere tutti i mimetype.
`GetURIAppsCanWrite()`	Ritorna una lista di URI di applicazioni in grado di modificare il mimetype.
`GetURIAppsCanWriteAll()`	Ritorna una lista di URI di applicazioni in grado di modificare tutti i mimetype.

Tabella 4.4: Classe MimetypeManager

Classe `MimetypeManager`	
Classe gestore degli oggetti Mimetype. Li raccoglie così da poter risparmiare memoria (un solo oggetto per ogni tipo di mimetype), rendere più scorrevole il codice e più potenti le query.	
Metodo	**Commento**
`init(model)`	Costruttore. Viene passato in ingresso l'oggetto RDF.Model della libreria librdf rappresentante l'handler per la base semantica.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`AddMimetype(mimetype, complexityLevel)`	Viene passata in ingresso la stringa del mimetype da gestire (ad esempio 'text/-plain'). Dalla stringa viene istanziato il relativo oggetto Mimetype qualora non sia già presente nel gestore. Può inoltre essere passato in input il valore complexityLevel (da 0 a 5) stante ad indicare il livello di complessità desiderato per le applicazioni (con 0 si vuole indicare che vanno prese in considerazione tutte le applicazioni a prescindere dal loro livello di complessità). Vengono inoltre recuperate le URI e creati gli oggetti Application delle applicazioni in grado di leggere e/o scrivere il/i mimetype.
`GetAppsCanRead()`	Ritorna una lista delle applicazioni (oggetti Application) in grado di leggere tutti i mimetype presenti nell'oggetto (MimetypeManager).
`GetGUIAppsCanRead()`	Ritorna una lista delle applicazioni dotate di GUI (oggetti Application) in grado di leggere tutti i mimetype presenti nell'oggetto (MimetypeManager).
`GetAppsCanWrite()`	Ritorna una lista delle applicazioni (oggetti Application) in grado di modificare tutti i mimetype presenti nell'oggetto (MimetypeManager).
`GetGUIAppsCanWrite()`	Ritorna una lista delle applicazioni dotate di GUI (oggetti Application) in grado di modificare tutti i mimetype presenti nell'oggetto (MimetypeManager).
Continua nella pagina successiva	

Continua dalla pagina precedente	
`GetAppsCanHandle()`	Ritorna una lista delle applicazioni (oggetti Application) in grado di operare su tutti i mimetype presenti nell'oggetto (MimetypeManager). In pratica si tratta di una unione tra la lista delle applicazioni in grado di leggere e quelle in grado di modificare i mimetype.
`GetGUIAppsCanHandle()`	Ritorna una lista delle applicazioni dotate di GUI (oggetti Application) in grado di operare su tutti i mimetype presenti nell'oggetto (MimetypeManager). In pratica si tratta di una unione tra la lista delle applicazioni in grado di leggere e quelle in grado di modificare i mimetype.
`GetMimetypeObject( mimetype)`	Ritorna l'oggetto Mimetype della stringa mimetype (ad esempio 'text/plain') passata come input. Il mimetype deve essere presente (quindi aggiunto utilizzando il metodo AddMimetype) nel MimetypeManager. Qualora non sia presente viene restituito None.
`GetMimetypesList()`	Ritorna una lista di stringhe rappresentanti i mimetype gestiti dall'oggetto (ad esempio ['text/plain', 'text/rtf']).
`GetOperations()`	Ritorna una lista di oggetti Operation rappresentante tutte le possibili operazioni effettuabili sui mimetype gestiti dal MimetypeManager dalle applicazioni presenti nel sistema.
`ChangeComplexityLevel( level)`	Cambia l'attuale livello di complessità. Ricarica quindi i dati relativi ai mimetype tenendo presente il nuovo livello di complessità richiesto (un numero da 0 a 5).

Tabella 4.5: Classe Category

Classe `Category`	
Classe rappresentante una categoria.	
Metodo	**Commento**
`init(uri, model)`	Costruttore. Viene passato in ingresso l'URI della categoria da rappresentare e l'oggetto RDF.Model della libreria librdf rappresentante l'handler per la base semantica.
`GetName()`	Ritorna il nome della categoria.

Tabella 4.6: Classe CategoriesManager

Classe `CategoriesManager`	
Classe gestore delle categorie delle operazioni. Serve a risparmiare memoria (un'unico oggetto Category per ogni categoria e non uno per ogni oggetto Operation) e per rendere più leggibile il codice. Offre metodi per aggiungere categorie e per recuperare informazioni riguardo al loro ordine gerarchico.	
Metodo	**Commento**
`init(model)`	Costruttore. Viene passato in ingresso l'oggetto RDF.Model della libreria librdf rappresentante l'handler per la base semantica.
`AddCategory(uriCat)`	Aggiunge al dizionario la categoria passata in ingresso (sotto forma di URI).
`GetRootCategories()`	Ritorna una lista contenente le categorie radice.
`GetChildsOf(category)`	Ritorna una lista contenente le categorie figlie della categoria passata in ingresso (un oggetto Category).

Tabella 4.7: Classe Operation

Classe `Operation`	
Classe contenente informazioni riguardo ad una singola operazione (nome, mimetype di ritorno, se si tratta di una operazione di gruppo, help, categoria di appartenenza e le applicazioni in grado di compierla).	
Metodo	**Commento**
`init(uri, model)`	Costruttore. Viene passato in ingresso l'URI dell'operazione da rappresentare e l'oggetto RDF.Model della libreria librdf rappresentante l'handler per la base semantica.
`GetName()`	Ritorna il nome dell'operazione.
`GetResultMimetype()`	Ritorna una lista di stringhe dei mimetype in ritorno all'operazione.
`GetShortComment()`	Ritorna l'eventuale help di commento all'operazione.
`GetURICategory()`	Ritorna la URI dell'eventuale categoria di appartenenza.
`IsGroupOperation()`	Ritorna true se l'operazione è una operazione di gruppo, false altrimenti.

Tabella 4.8: Classe Parameter

Classe `Parameter`	
Classe rappresentante un parametro di una operazione. Contiene i dati che lo riguardano (opzione di linea di comando, nome della label per l'interfaccia utente, tipo del parametro e eventuale help). La caratterizza il metodo __cmp__ necessario per poter ordinare i parametri in basa al parameterOrder. (vedi sez. 4.1.11)	
Metodo	**Commento**
`init()`	Costruttore. Inizializza alcune variabili dell'oggetto.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`__cmp__(other)`	Metodo necessario per poter ordinare una lista di oggetti Parameter in base al loro parameterOrder (vedi sez. 4.1.11).

Tabella 4.9: Classe Application

Classe `Application`	
Classe contenente i dati relativi ad una applicazione (nome, path di installazione, versione, licenza, prezzo...). Permette inoltre di ottenere la lista dei parametri necessari all'esecuzione di una operazione.	
Metodo	**Commento**
`init(uri, model)`	Costruttore. Viene passato in ingresso l'URI dell'applicazione da rappresentare e l'oggetto RDF.Model della libreria librdf rappresentante l'handler per la base semantica.
`GetName()`	Ritorna il nome dell'applicazione.
`GetInstallationPath()`	Ritorna il path in cui è stata installata l'applicazione.
`GetExecCommand()`	Ritorna il nome dell'eseguibile dell'applicazione.
`GetFullPath()`	Ritorna il path completodell'eseguibile dell'applicazione.
`GetVersion()`	Ritorna il numero di versione.
`GetLicense()`	Ritorna una stringa indicativa del tipo di licenza (commerciale, freeware, open-source, etc.).
`GetPrice()`	Ritorna il prezzo dell'applicativo.
`GetWebURL()`	Ritorna l'indirizzo web dell'applicazione.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`GetParameters(operation)`	Ritorna una lista di oggetti Parameter rappresentanti i parametri necessari all'esecuzione da parte dell'applicazione dell'operazione passata in input (un oggetto Operation).
`HasGUI()`	Ritorna True se l'applicazione è dotata di interfaccia utente, False altrimenti.

4.1.4 Le classi dell'interfaccia utente

Di seguito vengono riportate, in forma tabellare, le classi, con la spiegazione dei relativi metodi, necessarie alla creazione dell'interfaccia utente dell'applicazione.

Tabella 4.10: Classe Flusso

Classe `Flusso`	
Si tratta della classe di partenza dell'applicazione. Ha il compito di creare il database semantico processando i file RDF disponibili nel sistema e di raccogliere i dati passati attraverso la linea di comando. È derivata dalla classe wxApp della libreria wxWidgets.	
Metodo	**Commento**
`OnInit(uri, model)`	Crea il database semantico e recupera i parametri passati a linea di comando (ossia i file in input al flusso di operazioni).

Tabella 4.11: Classe MainFrame

Classe `MainFrame`	
Crea menu e statusbar ed è il contenitore per i pannelli dell'interfaccia utente (SummaryPanel, OperationPanel e OutputPanel, vedi sez. 4.1.7). Ha inoltre il compito di capire il mimetype dei file passati in ingresso tramite linea di comando alla classe Flusso e di predisporre i bottoni per aggiungere operazioni e per far "partire" il flusso. (La classe è derivata da wxFrame del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, ID, title, fileStart, model)`	Costruttore. Iniziallizza le variabili, raccoglie informazioni sui mimetype e prepara l'interfaccia grafica. In input ha bisogno di: **parent** handle alla finestra padre; **ID** wxId della finestra che si andrà a creare; **title** Titolo della finestra che si andrà a creare; **fileStart** lista dei file in input (path completo); **model** handle all'oggetto RDF.Model.
`OnOpen(event)`	Gestisce l'evento generato dal menu File-Open. Carica un flusso salvato dal metodo OnSave.
`OnSave(event)`	Gestisce l'evento generato dal menu File-Save. Salva un flusso su file.
`OnPreferences(event)`	Risponde all'evento generato dal menu File-Preferenze.... Attualmente permette di cambiare la directory in cui si trovano i file RDF.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`OnShowApps(event)`	Gestisce l'evento generato dal menu qualora si voglia abilitare o disabilitare la visione dei combo-box relativi alle applicazioni nei pannelli OperationPanel. Di default sono invisibili.
`OnShowComplexity(event)`	Gestisce l'evento generato dal menu qualora si voglia abilitare o disabilitare la visione dello slider che permette la scelta della difficoltà delle applicazioni. Di default è invisibile e la difficoltà è settata ad un valore medio pari a 2.
`OnAddOperation(event)`	Gestisce l'evento generato dal pulsante "Aggiungi operazione" aggiungendo un nuovo pannello OperationPanel. Demanda il tutto al metodo AddOperationPanel.
`AddOperationPanel()`	Esegue le operazioni necessarie all'aggiunta di un nuovo pannello OperationPanel.
`OnChangedOperation(event)`	Gestisce l'evento generato dalla selezione di una operazione dal combo delle operazioni in qualche OperationPanel. È infatti importante capire se l'operazione scelta ha un mimetype di ritorno o meno così da poter creare o distruggere l'OutputPanel che gestisce i nomi degli eventuali file in uscita dal flusso. Inoltre se viene cambiata un'operazione che si trova nel mezzo del flusso di operazioni e quest'ultima ha un mime-type di ritorno non compatibile con le operazioni successive allora queste ultime dovranno venir rimosse dal flusso (dopo conferma da parte dell'utente).
`OnRemoveOperation(event)`	Gestisce l'evento generato da qualche pannello OperationPanel nel caso richieda di essere tolto dalla lista delle operazioni.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`OnStartFlow(event)`	Percorre l'elenco dei pannelli OperationPanel eseguendo l'operazione richiesta. I risultati delle operazioni sono salvati su file temporanei. Il processo non parte nel caso in cui non sia specificato un nome per il file risultante dal flusso di operazioni (qualora sia previsto). Tale nome è gestito dalla classe OutputPanel.
`Adjust()`	Ridimensiona e aggiusta le posizioni dei controlli della finestra.
`TimeToQuit(event)`	Gestisce l'evento generato dal menu per uscire dalla applicazione.

Tabella 4.12: Classe OpScrolledWindow

Classe `OpScrolledWindow`	
Contiene i pannelli OperationPanel in un'unica finestra scorrevole (in senso verticale) così che un'eventuale flusso di operazioni molto lungo non occupi più schermo di quello disponibile. (La classe è derivata da wxScrolledWindow del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent)`	Costruttore, richiede la finestra padre come parametro in input (in questo caso `MainFrame`)
`AddOperationPanel(op)`	Aggiunge alla finestra l'oggetto OperationPanel passato come parametro in ingresso.
`RemoveOperationPanel(op)`	Rimuove dalla finestra l'oggetto OperationPanel passato come parametro in ingresso.

Tabella 4.13: Classe SummaryPanel

Classe `SummaryPanel`	
Mostra a video le informazioni riguardo al mimetype del/dei file di partenza e le applicazioni in grado di manipolarlo. Dispone inoltre del bottone per aprire i/il file con una applicazione scelta dall'utente. (La classe è derivata da wxPanel del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, fileName, mimetype, model)`	Costruttore. In input ha bisogno di: **parent** la finestra padre (in questo caso è MainFrame); **fileName** lista dei file in input (path completo); **mimetype** lista di stringhe dei mimetype dei file in input (deve essere nello stesso ordine della lista fileName) **model** handle all'oggetto RDF.Model.
`CreateControls()`	Crea i controlli dell'interfaccia utente.
`FillControls()`	Riempie con i relativi dati i controlli dell'interfaccia utente.
`ShowOpenWith(value)`	Mostra o nasconde il bottone "Apri con" a seconda del valore (True o False) del parametro in ingresso.
`ShowApps(value)`	Mostra o nasconde il combo delle applicazioni a seconda del valore (True o False) del parametro in ingresso.
`ShowComplexityLevel(value)`	Mostra o nasconde lo slider per scegliere il grado di complessità delle applicazioni a seconda del valore (True o False) del parametro in ingresso.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`OnComplexity(event)`	Metodo richiamato in risposta all'evento generato dallo slider della complessità quando cambia valore. Vanno quindi recuperate nuovamente le applicazioni in grado di gestire i mimetype e apportate le modifiche all'interfaccia utente.
`OnOpenWith(event)`	Gestisce l'evento generato dal pulsante "Apri con" lanciando l'applicazione selezionata nel relativo combo box.
`GetMimetypeManager()`	Ritorna l'oggetto MimetypeManager.

Tabella 4.14: Classe OperationPanel

Classe `OperationPanel`	
Rappresenta una operazione applicabile sul/sui mimetype correnti da parte delle applicazioni presenti nel sistema. (La classe è derivata da wxPanel del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, mimetypeManager, model)`	Costruttore. In input ha bisogno di: **parent** la finestra padre (in questo caso è MainFrame); **mimetypeManager** oggetto MimetypeManager per la gestione dei mimetype del pannello; **model** handle all'oggetto RDF.Model.
`StartPanel()`	Richiama i metodi per riempire i controlli del pannello (comboCox eccetera). Non viene fatto dal costruttore per permettere agli oggetti padre di inizializzare gli eventhandler per gli eventi RemoveOperation e ChangedOperation.
`FillControls()`	Riempie i controlli del pannello con i dati relativi alle operazioni e alle categorie.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`GetResultMimetypes()`	Ritorna la lista dei mimetype di ritorno dall'operazione (lista di stringhe, ad esempio ['text/plain', 'image/jpeg']. None se l'operazione non prevede file in output.
`GetInputMimes()`	Ritorna la lista dei mimetype in ingresso al pannello.
`ShowApplications(value)`	Mostra o nasconde (in base al boolean value in ingresso) il combo-box delle applicazioni.
`OnButtonRemoveOperation(event)`	Gestisce l'evento dato dal bottone "Rimuovi questa operazione". Il metodo genere a sua volta un evento RemoveOperation che verrà gestito dalla finestra padre (MainFrame) rimuovendo il pannello dal flusso.
`OnButtonCategory(event)`	Gestisce l'evento generato dalla selezione di una categoria da uno dei combo delle categorie. Demanda in realtà la gestione al metodo ProcessCategory.
`ProcessCategory(combo)`	Esegue le operazioni necessarie qualora venga scelta una categoria da uno dei combo delle categorie. In input viene passato l'oggetto wxChoice generatore dell'evento.
`OnButtonHelpOperation(event)`	Risponde all'evento generato dalla pressione del bottone che permette di ottenere maggiori informazioni riguardo all'operazione scelta.
`OnSelectedOperation(event)`	Gestisce l'evento generato dalla selezione di una operazione dal combo delle operazioni. Demanda in realtà la gestione al metodo ProcessOperation.
`ProcessOperation()`	Esegue le operazioni necessarie qualora venga scelta una operazione nel combo delle operazioni.
`AddOperation(uriOp)`	Aggiunge l'operazione uriOp al pannello.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`GetCurrentOperation()`	Ritorna l'attuale operazione scelta. None se non ne è selezionata nessuna.
`DrawParameters()`	Gestisce i parametri dell'operazione scelta disegnando coerentemente l'interfaccia utente.
`OnSelectedApplication( event)`	Gestisce l'evento generato dalla selezione di una applicazione dal combo delle applicazioni. In sostanza richiama il metodo DrawParameters.

Tabella 4.15: Classe TextCtrlValidator

Classe `TextCtrlValidator`	
Ha il compito di verificare che il testo immesso in un text-box sia valido, ossia che sia composto di soli caratteri, di soli numeri o di entrambi. La verifica viene eseguita ad ogni carattere immesso. (La classe è derivata da wxPyValidator del pacchetto wxWidgets).	
Metodo	**Commento**
`init(flag)`	Costruttore. Richiede in input un flag indicante il tipo di testo da accettare nel controllo (ALPHA_ONLY per caratteri alfabetici, DIGIT_ONLY per valori numerici e ALPHA_DIGIT per entrambi)
`Clone()`	Metodo standard Clone. Ogni classe derivata da wxPyValidator deve implementare questo metodo.
`Validate(ctrl)`	Controlla se il testo del controllo ctrl è valido.
`OnChar(event)`	Verifica il carattere apppena immesso.

Tabella 4.16: Classe OutputPanel

Classe `OutputPanel`	
Gestisce i nomi dei file in output al flusso di operazioni. Se il file in output è unico risulta sufficiente richiedere un unico nome, se i file sono molti viene richiesto un prefisso comune a cui verranno aggiunti dei suffissi del tipo _0001, _0002, _00xy. (La classe è derivata da wxPanel del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, justOne)`	Costruttore. In input ha bisogno dei parametri `parent`, ossia la wxWindow padre e `justOne`, booleano posto a true se il file in uscita dal flusso è unico, false altrimenti.
`OnBrowseButton(event)`	Gestisce l'evento generato dal pulsante "Sfoglia..." aprendo una finestra di dialogo per scegliere più agevolmente nome e path del/dei file in output.
`AddLabels()`	Aggiunge all'interfaccia utente delle etichette che forniscono una frase di help qualora i file in output previsti dal flusso siano più di uno.
`RemoveLabels()`	Rimuove i campi aggiunti dal metodo AddLabels.
`SetJustOne(value)`	Setta il valore di justOne (vedi costruttore) in accordo con il parametro value. Di conseguenza cambia l'interfaccia utente (se necessario).
`IsOk()`	Ritorna true se è stato immesso un nome per il/i file in output, false altrimenti.
`Reset()`	Resetta il conteggio del suffisso per i file in output.
`GetNextName()`	Ritorna il prossimo nome da utilizzare come file in output.

Tabella 4.17: Classe GaugePanel

Classe `GaugePanel`	
Rappresenta un pannello dotato di progress bar indicante l'andamento del flusso di operazioni. (La classe è derivata da wxPanel del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, num_op)`	Costruttore. In input ha bisogno dei parametri `parent`, ossia la wxWindow padre e `num_op`, intero indicante il numero di operazioni presenti nel flusso.
`Reset()`	Resetta l'interfaccia.
`Step()`	Fa fare un passo alla progress bar.

Tabella 4.18: Classe RemoveOperation

Classe `RemoveOperation`	
Classe evento creata per notificare altre finestre della pressione del tasto "Rimuovi questa operazione" in un OperationPanel. In questo modo la classe MainFrame può intercettare l'evento e rimuovere i pannelli non più necessari. (La classe è derivata da wxPyCommandEvent del pacchetto wxPython).	
Metodo	**Commento**
`init(windowID, obj)`	Il costruttore necessita del wxId dell'oggetto wx a cui è associato e dell'oggetto OperationPanel generatore dell'evento.
`Clone()`	Crea una copia dell'oggetto.
`GetEventObject()`	Ritorna l'oggetto OperationPanel generatore dell'evento.

Tabella 4.19: Classe ChangedOperation

Classe `ChangedOperation`	
Classe evento creata per notificare altre finestre dell'avvenuta scelta di una operazione nel combo-box delle operazioni in un OperationPanel. La particolarità sta nel metodo `GetResultMimes` che torna il/i mimetypes in un uscita dall'operazione selezionata. (La classe è derivata da wxPyCommandEvent del pacchetto wxPython).	
Metodo	**Commento**
`init(windowID, obj)`	Il costruttore necessita del wxId dell'oggetto wx a cui è associato e dell'oggetto Operation associato all'evento.
`Clone()`	Crea una copia dell'oggetto.
`GetEventObject()`	Ritorna l'oggetto OperationPanel generatore dell'evento.
`GetResultMimes()`	Ritorna la lista dei mimetype risultato.
`IsGroupOperation()`	Agisce come il metodo `IsGroupOperation` della classe `Operation`.

Tabella 4.20: Classe EditMimesDialog

Classe `EditMimesDialog`	
Si tratta di una finestra di dialogo richiamata qualora il sistema non riesca a risalire al mimetype di qualche file in input. In tal modo si da la possibilità all'utente di specificarlo manualmente. (La classe è derivata da wxDialog del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, mimetypes)`	Costruttore. In input ha bisogno di `parent`, finestra padre e `mimetypes`, dizionario di stringhe dei mimetype (supposti) dei file in input, ad esempio {'pippo.txt':'text/plain', 'pluto.xyz':'Ignoto'}.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`OnButtonOk(event)`	Gestore dell'evento generato dalla pressione del bottone Ok.
`GetMimetypes()`	Ritorna la lista dei mimetype con le eventuali modifiche dell'utente (ad esempio ['text/plain', 'text/rtf']).

4.1.5 Grafico UML

In figura 4.5 è illustrato il grafico UML delle relazioni tra le classi dell'applicazione Sflow. Sono state riportate sia le classi del modello (evidenziate utilizzando il colore blu), sia le classi dell'interfaccia utente (evidenziate utilizzando il colore arancio). Sono state omesse per chiarezza sia le relazioni tra le classi dell'interfaccia utente e le classi della libreria wxWidgets che le relazioni tra le classi del modello e quelle della libreria Redland. Mancano inoltre la descrizione di proprietà e metodi delle classi, sia perché già illustrate nelle sezioni precedenti, sia per non appesantire inutilmente il grafico.

4.1.6 Adattamento di applicazioni esistenti al modello

Nel paragrafo 3.4.4 viene illustrato come fare in modo che l'applicazione `convert`, presente nel pacchetto ImageMagick, possa descrivere il concetto di operazione di ridimensionamento di un'immagine e di come poterla eseguire utilizzando una certa sintassi. In realtà l'applicazione `convert` non segue le linee guida dettate in 3.4.1 per quanto riguarda la sintassi da utilizzare nella specifica di parametri a linea di comando. Volendo, ad esempio, convertire l'immagine `pippo.jpg` in bianco e nero si dovrebbe dare il seguente comando:
`convert pippo.jpg -monochrome pippo_bn.jpg`
Come si vede, il file in uscita `pippo_bn.jpg` non è stato introdotto dalla stringa `--output` e l'opzione `monochrome` è stata introdotta da un solo trattino al posto dei due richiesti, pertanto l'applicazione Sflow non sarebbe in grado di utilizzare correttamente il programma `convert`.

Per ovviare al problema si possono facilmente implementare dei semplici programmi aventi il compito di *riordinare* i paramentri a linea di comando

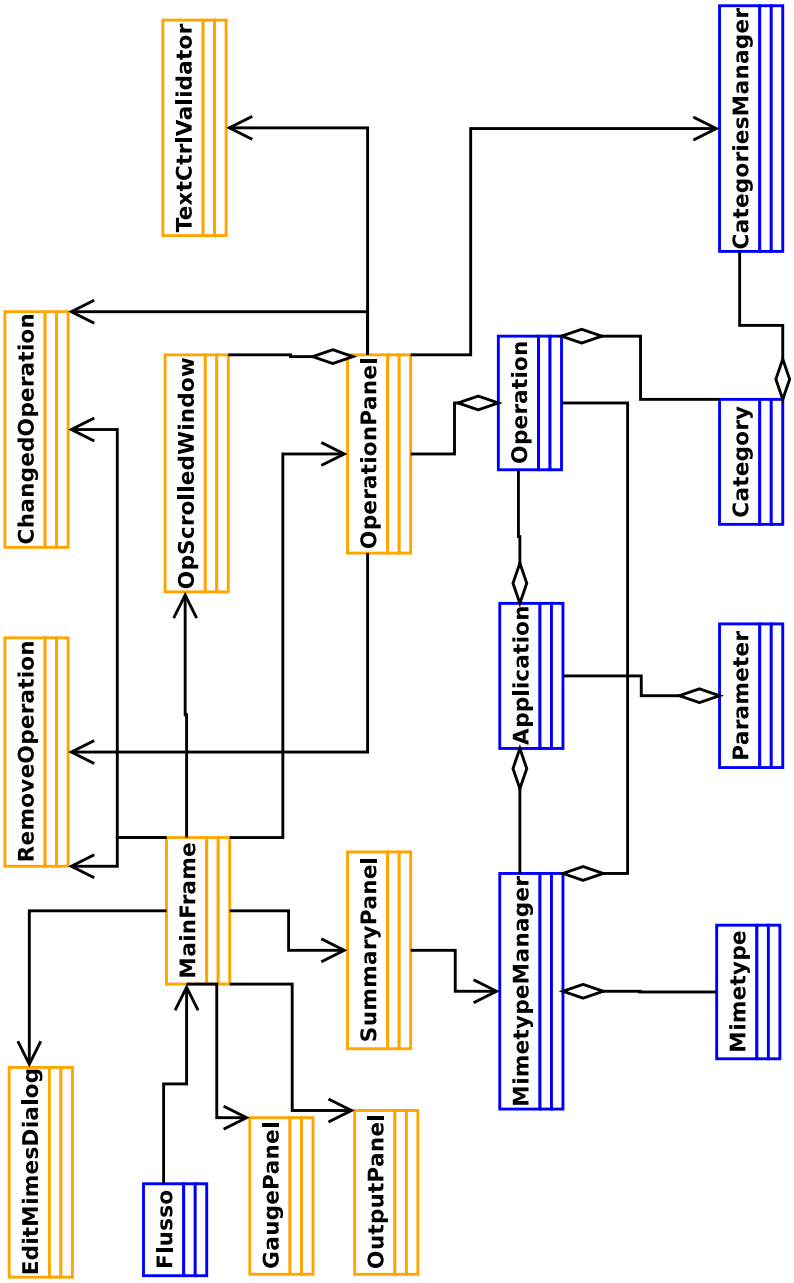

Figura 4.5: Diagramma UML delle classi di Sflow

così da renderli compatibili con quanto definito in 3.4.1. Ad esempio, utilizzando Python come linguaggio di programmazione, si possono adattare alcune operazioni di `convert` nel modo seguente:

```
#!/usr/bin/python
import os
import sys
from optparse import OptionParser

class convert_wrp:
    def __init__(self):
        print "Convert Wrapper"
        self.parser = OptionParser()

        self.options_list = ["--rotate", "--flop", "--monochrome"]

        self.parser.add_option("--rotate", action="store",
                               type="string", dest="rotate")
        self.parser.add_option("--flop", action="store_true",
                               dest="flop")
        self.parser.add_option("--monochrome", action="store_true",
                               dest="monochrome")

        self.parser.add_option("-o", "--output", action="append",
                               type="string", dest="outputFile")

        (self.options, self.args) = self.parser.parse_args()


    def performOperation(self):
        command = "/usr/bin/convert"

        lParams = []
        lParams.append(command)

        for inFile in self.args:
            lParams.append(inFile)

        if self.options.rotate != None:
            lParams.append("-rotate")
            lParams.append(self.options.rotate)
        if self.options.flop:
            lParams.append("-flop")
        if self.options.monochrome:
            lParams.append("-monochrome")

        for outFile in self.options.outputFile:
            lParams.append(outFile)

        result = os.spawnv(os.P_WAIT, command, lParams)
        return result
```

```
convert = convert_wrp()
sys.exit(convert.performOperation())
```

Questo piccolo wrapper è un semplice programma che accetta dei parametri in linea con lo standard dettato in 3.4.1 e converte alcune operazioni (rotazione, ribaltamento orizzontale e conversione in bianco e nero) nella sintassi accettata dal programma `convert`, richiamandolo in maniera opportuna. Fa quindi da tramite tra l'applicativo Sflow e `convert`.

Definito il wrapper è possibile ridefinire in maniera questa volta corretta quanto si era illustrato in 3.4.4. Si prenda ad esempio l'operazione di rotazione di un'immagine, possiamo descriverla nel seguente modo:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:mm="http://sflow.org/data/2005-06/mimetype#"
  xmlns:apps="http://sflow.org/schema/2005-06/applications#"
  xmlns:ops="http://sflow.org/schema/2005-06/operations#"
  xml:base="http://www.imagemagick.org/applications">


<ops:Operation rdf:ID="imageRotate">
  <ops:name>Ruota</ops:name>
  <ops:returnMimeType rdf:resource="http://sflow.org/data/2005-06/↩
      mimetype#sameasinput" />
  <ops:shortComment>Ruota un immagine di un certo numero di gradi</↩
      ops:shortComment>
</ops:Operation>


<apps:Application rdf:ID="convert">
  <apps:name>Convert</apps:name>
  <apps:installationPath>/usr/wrappers/</apps:installationPath>
  <apps:execCommand>convert_wrp.py</apps:execCommand>
  <apps:version>6.0</apps:version>
  <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      image_jpeg" />
  <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      image_png" />
  <apps:canPerform>
    <rdf:Bag>
      <rdf:li rdf:nodeID="_convertImageRotate" />
      </rdf:Bag>
  </apps:canPerform>
</apps:Application>

<rdf:Description rdf:nodeID="_convertImageRotate">
  <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
      mimetype#image_jpeg" />
```

Figura 4.6: Sflow: Operazione ridimensionamento immagine

```
  <apps:operation rdf:resource="#imageRotate" />
  <apps:parameters>
    <rdf:Seq>
      <rdf:li rdf:nodeID="_convertImageRotate_p1" />
    </rdf:Seq>
  </apps:parameters>
</rdf:Description>

<rdf:Description rdf:nodeID="_convertImageRotate_p1">
  <apps:commandOption>--rotate</apps:commandOption>
  <apps:parameterLabel>Angolo della rotazione:</apps:parameterLabel>
  <apps:parameterType rdf:resource="&xsd;decimal" />
  <apps:parameterHelp>Angolo in gradi centigradi della rotazione in ↩
      senso orario</apps:parameterHelp>
</rdf:Description>
</rdf:RDF>
```

Avendo cura di definire delle altre operazioni e di categorizzarle in maniera opportuna (vedi sez. 3.4.2) possiamo richiamare l'applicazione Sflow passandole a linea di comando il path dell'immagine che vogliamo trattare. Premendo poi il pulsante "Aggiungi operazione" otterremo una schermata come in figura 4.6.

4.1.7 Design dell'interfaccia

Un grande punto di forza del modello descritto e del programma Sflow è quello di poter dotare, con poco sforzo, di una interfaccia utente grafica tutti quegli applicativi nati per essere utilizzati a linea di comando. E per di più l'interfaccia è in qualche modo *indipendente* dall'applicativo in quanto fa di fatto parte solamente del programma Sflow. Questo comporta minor confusione all'utente, incrementandone la produttività e diminuendone la frustrazione data dalla diversità di interfaccia in funzione delle applicazioni.

È stata quindi posta molta enfasi anche nello studio dell'interfaccia utente di Sflow. Requisiti fondamentali sono stati la chiarezza, la semplicità e l'evitare il più possibile casi di *information overload*.

In sostanza l'interfaccia si sviluppa in verticale ed è stata divisa in tre sezioni principali che verranno illustrate singolarmente nelle successive sezioni:

- Una prima sezione di "riassunto" in cui vengono presentati il nome ed il tipo dei documenti in ingresso e le applicazioni in grado di gestirlo.
- La parte centrale è occupata dai diversi *pannelli operazione*. In sostanza dei moduli, ognuno dei quali contiene tutte le informazioni riguardanti una operazione del flusso.
- Nel caso sia necessario fornire dei nomi per eventuali file in output il tutto viene gestito dall'ultimo pannello, denominato OutputPanel.

Inoltre sono presenti un piccolo pannello con i pulsanti per aggiungere operazioni al flusso e farlo partire ed i classici menu e status bar (vedi figura 4.7).

SummaryPanel

In questo pannello (vedi figura 4.8) sono riassunte alcune informazioni riguardo al/ai file in ingresso al flusso. Viene riportato il nome, il mime-type e una lista, sotto forma di combo-box, di applicazioni in grado di leggere o modificare il file in questione. Scelta un'applicazione è possibile richiamarla premendo il pulsante "Apri con".

OperationPanel

Il compito di OperationPanel (vedi figura 4.9) è quello di poter far scegliere all'utente una operazione del flusso. Al principio (se si tratta cioè della prima

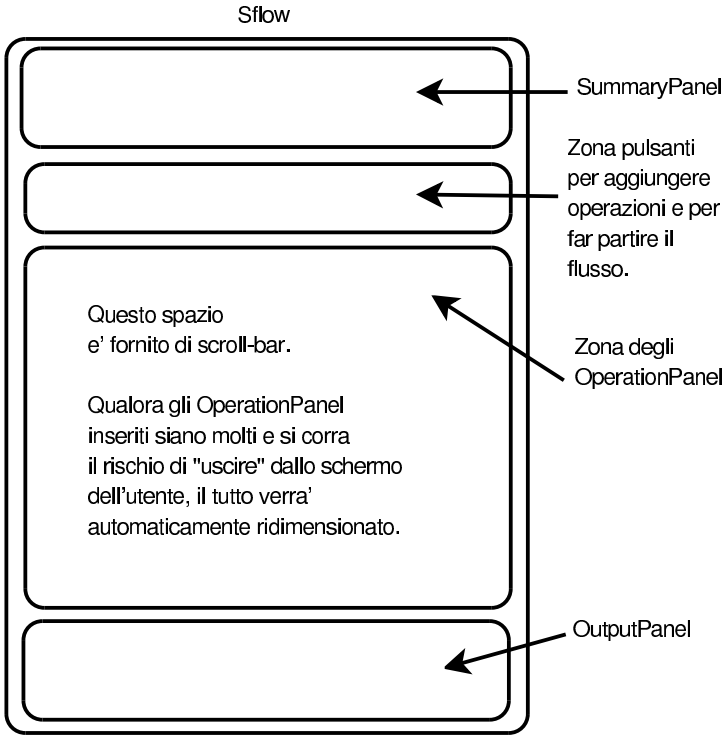

Figura 4.7: Schema interfaccia utente

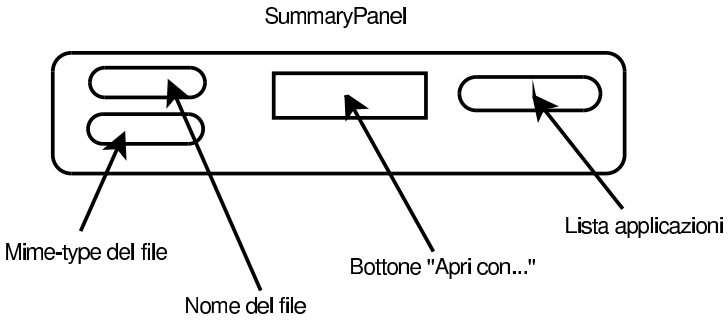

Figura 4.8: Schema interfaccia utente di SummaryPanel

operazione) dovrà fornire la lista delle operazioni effettuabili sui documenti in ingresso da parte delle applicazioni installate nel sistema. Per i pannelli successivi dovrà fare riferimento ai mime-type in uscita dal pannello precedente. Da notare che le operazioni possono essere dotate di categoria e le categorie sono organizzate in modo gerarchico. Per tal motivo le categorie sono presentate all'interno di combo-box. Ogni combo-box contiene le categorie di uno stesso livello e le eventuali sottocategorie appariranno via via in nuovi combo-box in base alle scelte effettuate dall'utente.

Compito di questo pannello è anche quello di richiedere all'utente eventuali parametri richiesti dall'operazione scelta. Attualmente sono gestiti da Sflow parametri di tipo `xsd:decimal`, `xsd:string` e `xsd:boolean` ossia valori numerici, stringhe o valori booleani.

All'utente sono forniti due gradi di help. Il primo, molto semplice, appare sotto forma di *tip* e fa riferimento alle proprietà `shortComment` e `parameterHelp` delle classi `Operation` e `Application`.
Il secondo livello di help è fornito sotto forma di pagina (X)HTML e apparirà in seguito alla pressione del tasto raffigurante un punto di domanda di fianco all'operazione (tale tasto apparirà solo se la proprietà `URLComment` è definita per l'operazione in questione).

OutputPanel

Sarà quasi sempre necessario fornire un nome per i documenti in uscita al flusso. Compito di OutputPanel (vedi figura 4.10) è quello di fornire l'interfaccia utente per permettere all'utente di fornire i nomi ai documenti. Nel caso il documento in uscita sia unico, OutputPanel si riduce ad un semplice campo di testo in cui inserire il nome. Nel caso, invece, in cui i documenti siano più di uno verrà richiesto di fornire un prefisso comune a tutti i file a cui verrà poi aggiunto automaticamente un suffisso numerico crescente.

È sempre presente anche un pulsante "Sfoglia..." la cui pressione farà apparire una finestra di dialogo avente lo scopo di rendere più semplice per l'utente la scelta del path in cui andare a salvare i documenti in uscita dal flusso di operazioni.

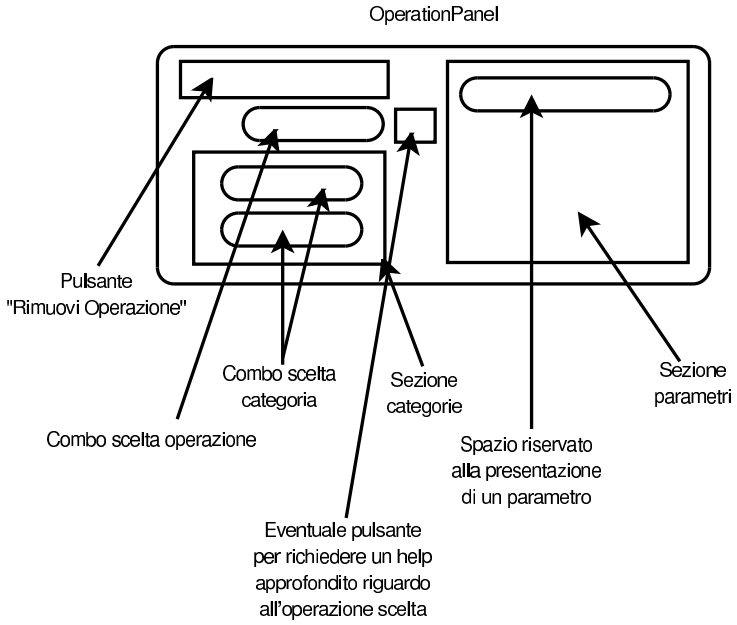

Figura 4.9: Schema interfaccia utente di OperationPanel

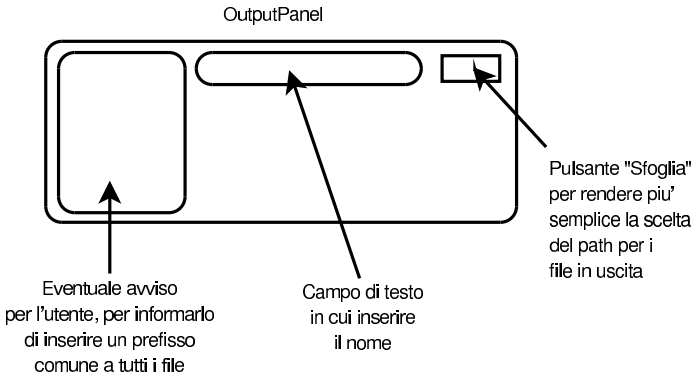

Figura 4.10: Schema interfaccia utente di OutputPanel

4.1.8 Esempi di flussi

Documenti PDF

Supponiamo di avere un documento Pdf e di doverlo modificare e spedire via posta elettronica ad una persona di nostra conoscenza. La prima operazione da fare sarà quella di convertire il documento originale in un formato modificabile (in quanto i documenti Pdf non lo sono). Potremmo quindi apportare le modifiche desiderate (ad esempio sostituire delle parole con delle altre) e quindi spedire il risultato via email (vedi figura 4.11).

In questo esempio Sflow ha utilizzato tre applicativi differenti:

- Un convertitore a testo semplice. In questo caso si è utilizzato `pdftotext`, ma era disponibile, ad esempio, anche `pdf2Ascii`.
- Il programma `sed` [36] che preso in input il documento di testo output di `pdftotext` ha sostituito i termini "pere" con "mele".
- Il semplice applicativo `sendemail` [37] che fornisce la funzionalità di spedire messaggi di posta elettronica con supporto per allegati.

Immagini

In questo esempio partiamo da una serie di immagini in formato Jpeg, le riduciamo di dimensione (in particolare le portiamo ad una larghezza di 300 pixel con l'altezza modificata automaticamente in modo da mantenere le proporzioni) e applichiamo una nota (una stringa) su ognuna di esse (cosa che può risultare utile se vogliamo pubblicare delle immagini su internet ma vogliamo che siano "marchiate" in modo semplice come nostre). Vedi figura 4.12

In questo caso Sflow ha utilizzato le funzionalità di un unico applicativo esterno ossia `convert` della suite ImageMagick [30].

File audio

Supponiamo di avere una canzone in formato Mp3 e di volerla segnalare ad un nostro amico via posta elettronica in quanto abita lontano da noi. Non si vuole però spedire il file completo, in quanto piuttosto pesante e non siamo dotati di banda larga. Risulterebbe quindi utile tagliare un pezzo di canzone e spedire quest'ultimo (vedi figura 4.13).

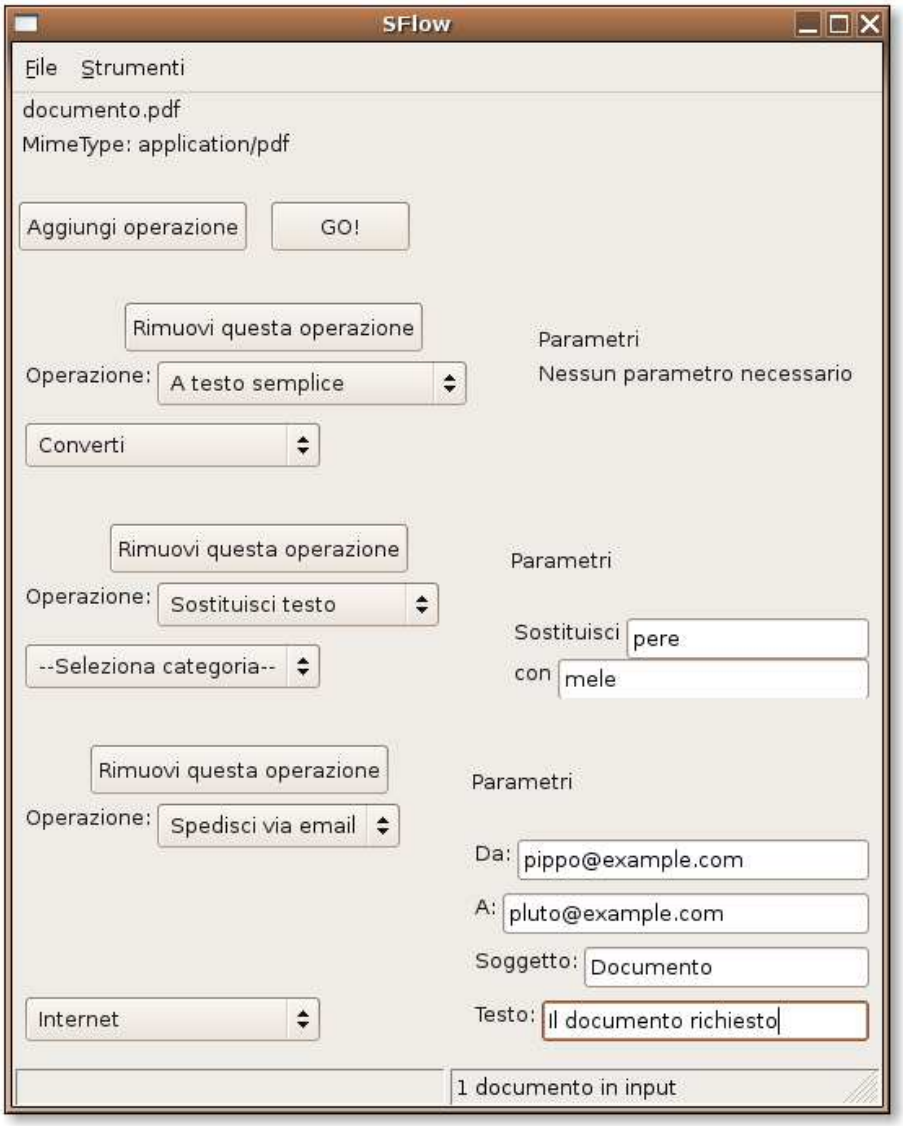

Figura 4.11: Esempio di flusso a partire da un documento Pdf

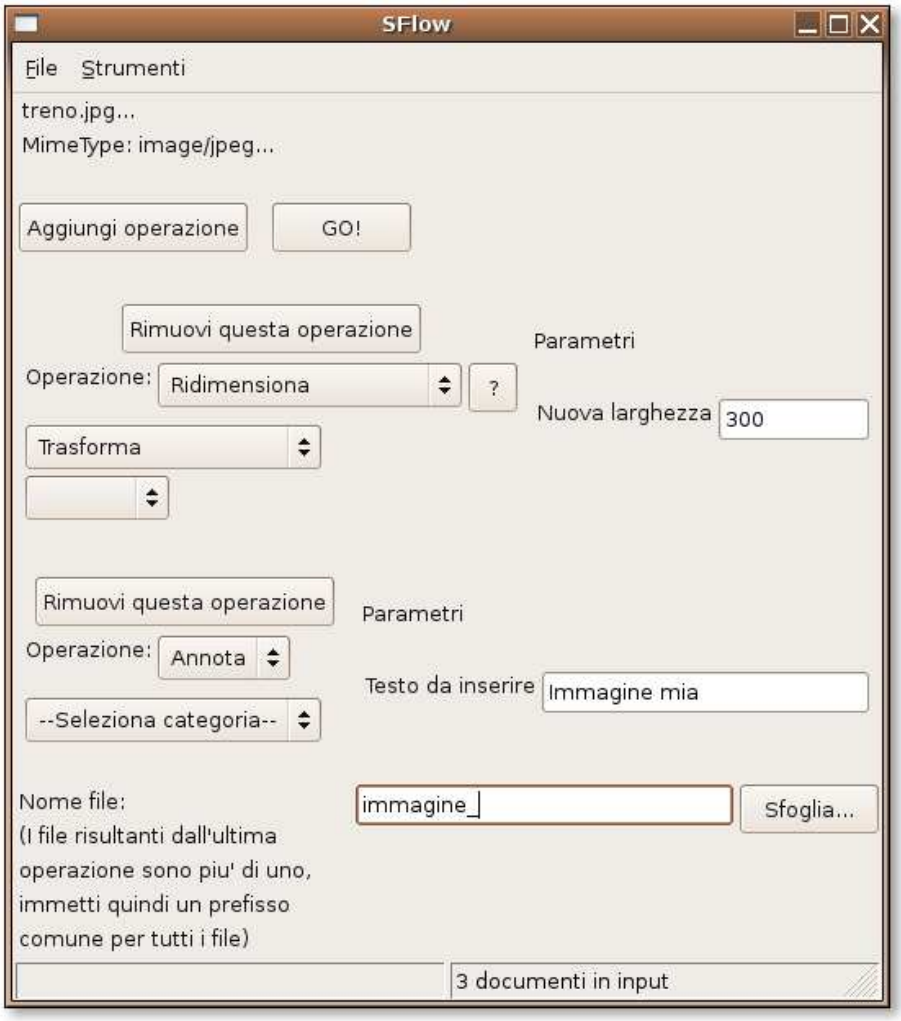

Figura 4.12: Esempio di flusso a partire da varie immagini Jpeg

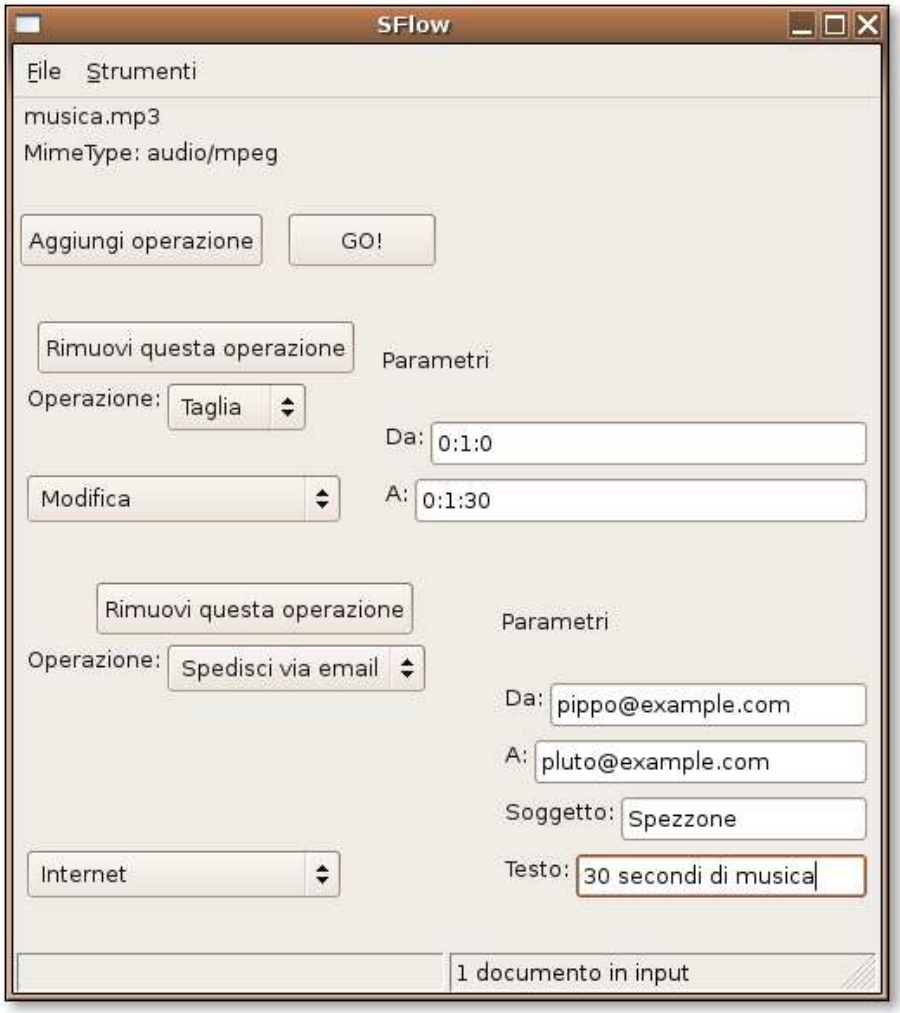

Figura 4.13: Esempio di flusso a partire da un documento Mp3

In questo esempio Sflow ha utilizzato due programmi esterni per compiere il flusso di lavoro:

- `qmp3cut` della suite Quelcom [33] per tagliare un pezzo dalla canzone originale.
- `sendemail` [37] per spedire il tutto via email.

`qmp3cut` ha una sintassi a linea di comando del tipo:
`qmp3cut -S <timeslice> outfile.mp3 infile.mp3`
dove `<timeslice>` indica i punti di inizio e fine del taglio che si desidera fare. Ogni punto va specificato tramite la sintassi `h:m:s` con `h` pari all'ora, `m` al minuto e `s` al secondo. Quindi una `timeslice` valida è una stringa del tipo `0:1:0-0:2:0`.

Anche in questo caso, come visto in 4.1.6, la sintassi della linea di comando di `qmp3cut` non è compatibile con quella definita in 3.4.1 per cui è stato necessario creare un piccolo script di conversione:

```
#!/usr/bin/python
import os
import sys
from optparse import OptionParser

class qmp3cut_wrp:
    def __init__(self):
        print "qMp3Cut Wrapper"
        self.parser = OptionParser()

        self.options_list = ["--start", "--end"]

        self.parser.add_option("--start", action="store",
                        type="string", dest="start")
        self.parser.add_option("--end", action="store",
                        type="string", dest="end")

        self.parser.add_option("-o", "--output", action="append",
                        type="string", dest="outputFile")

        (self.options, self.args) = self.parser.parse_args()


    def performOperation(self):
        command = "/usr/bin/qmp3cut"

        lParams = []
        lParams.append(command)

        if (self.options.start != None) and (self.options.end != None)↩
            :
```

```
            lParams.append("-S")
            lParams.append(self.options.start + "-" + self.options.end↩
                )

            lParams.append("-o")
            for outFile in self.options.outputFile:
                lParams.append(outFile)

        for inFile in self.args:
            lParams.append(inFile)

        result = os.spawnv(os.P_WAIT, command, lParams)
        return result


qmp3cut = qmp3cut_wrp()
sys.exit(qmp3cut.performOperation())
```

In questo modo Sflow userà una sintassi del tipo:
`qmp3cut_wrp.py infile.mp3 --start 0:1:0 --end 0:2:0 -o outfile.mp3`
che verrà tradotta in:
`qmp3cut -S 0:1:0-0:2:0 outfile.mp3 infile.mp3`

Creato il wrapper è stato possibile rendere pubbliche le informazioni riguardo a `qmp3cut` tramite il seguente documento RDF:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:mm="http://sflow.org/schema/2005-06/mimetypes#"
  xmlns:apps="http://sflow.org/schema/2005-06/applications#"
  xmlns:ops="http://sflow.org/schema/2005-06/operations#"
  xml:base="http://sflow.org/data/2005-06/application">

<apps:Application rdf:ID="qmp3cut">
  <apps:name>qMp3Cut</apps:name>
  <apps:installationPath>/usr/wrappers</apps:installationPath>
  <apps:execCommand>qmp3cut_wrp.py</apps:execCommand>
  <apps:version>0.4</apps:version>
  <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/mimetype#↩
      audio_mpeg" />

  <apps:canPerform>
    <rdf:Bag>
      <rdf:li rdf:nodeID="_mp3cut" />
    </rdf:Bag>
  </apps:canPerform>
</apps:Application>
```

```
<rdf:Description rdf:nodeID="_mp3cut">
  <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
      mimetype#audio_mpeg" />
  <apps:operation rdf:resource="http://sflow.org/data/2005-06/↩
      operation#audioCut" />
  <apps:parameters>
    <rdf:Seq>
      <rdf:li rdf:nodeID="_cut_p1" />
      <rdf:li rdf:nodeID="_cut_p2" />
    </rdf:Seq>
  </apps:parameters>
</rdf:Description>

<rdf:Description rdf:nodeID="_cut_p1">
  <apps:parameterOrder>1</apps:parameterOrder>
  <apps:commandOption>--start</apps:commandOption>
  <apps:parameterLabel>Da:</apps:parameterLabel>
  <apps:parameterType rdf:resource="&xsd;string" />
  <apps:parameterHelp>Punto di inizio del taglio (h:m:s)</↩
      apps:parameterHelp>
</rdf:Description>

<rdf:Description rdf:nodeID="_cut_p2">
  <apps:parameterOrder>2</apps:parameterOrder>
  <apps:commandOption>--end</apps:commandOption>
  <apps:parameterLabel>A:</apps:parameterLabel>
  <apps:parameterType rdf:resource="&xsd;string" />
  <apps:parameterHelp>Punto finale del taglio (h:m:s)</↩
      apps:parameterHelp>
</rdf:Description>

</rdf:RDF>
```

4.1.9 Altre caratteristiche

Specificare la fonte dei dati semantici

La prima volta che il programma Sflow è avviato, viene richiesto all'utente di specificare la directory in cui sono situati i file RDF che andranno a comporre la base semantica.
Tale directory potrà eventualmente essere modificata utilizzando l'opzione "Preferenze" del menu "File".

Figura 4.14: Sflow: scelta del grado di complessità

Scelta della complessità

Come descritto nella sezione 3.4.3 le applicazioni, e quindi le loro operazioni, possono essere categorizzate in base al loro livello di complessità utilizzando un numero intero da 1 a 5. Il programma Sflow gestisce queste informazioni attraverso una slide-bar (vedi figura 4.14) che rende possibile per l'utente scegliere il grado di difficoltà desiderato.
Normalmente la slide-bar è nascosta ed il livello di complessità è settato ad un valore medio pari a 2. Qualora l'utente volesse cambiarlo può agire sulla voce "Mostra complessità" del menu "Strumenti".

Scelta dell'applicativo

Può capitare che più appplicazioni siano in grado di eseguire la medesima operazione. In questi casi il programma Sflow si comporta scegliendo autonomamente una tra le possibili applicazioni in grado di compiere l'operazione in questione. Resta comunque possibile per l'utente poter scegliere esplicitamente a quale applicazione far compiere l'operazione. È infatti sufficiente agire sulla voce "Mostra applicazioni" del menu "Strumenti" per far apparire dei combo-box nei vari *pannelli operazione* da cui poter scegliere l'applicativo preferito (vedi figura 4.15).

Figura 4.15: Sflow: scelta esplicita dell'applicativo

Salvare e riutilizzare i flussi

Potrebbe risultare utile la possibilità di salvare i flussi di lavoro creati per poterli riutilizzare nuovamente in seguito senza la necessità di doverli ricreare da capo. Avendo, ad esempio, creato un flusso di operazioni che, avute delle immagini in ingresso, le debba ridimensionare, contrassegnare con una nota, comprimere in un unico file e spedire via mail e se si trattasse di una serie di operazioni che si vanno a ripetere molto spesso è conveniente salvarle; per fare ciò si può utilizzare la voce "Salva..." del menu "File".

Sarà poi possibile recuperare i flussi salvati agendo sulla voce "Apri..." del menu "File".

4.1.10 Cross-platform

Come descritto nel capitolo 2 l'applicativo Sflow è stato sviluppato con l'intento di essere *cross-platform*, ossia in grado di funzionare su diversi tipi di architettura e sistema operativo.

Nelle figure 4.16, 4.17 e 4.18 è illustrato lo stesso flusso di operazioni in

Figura 4.16: Sflow in ambiente Gnome

tre diversi sistemi operativi: Debian GNU/Linux in ambiente Gnome, Apple MacOS X 10.4.2 e Microsoft Windows XP.

4.1.11 Problemi

Nella realizzazione del prototipo sono emersi alcuni problemi di natura tecnica. In particolare alcuni sono dovuti a carenze della libreria utilizzata per il trattamento dei dati RDF e altri alla difficoltà di rendere pienamente cross-platform gli applicativi.
Verranno di seguito illustrati.

Lo "spazio" nel path

Attualmente nè i moduli specifici del linguaggio Python nè la libreria wxWidgets riescono a lavorare correttamente (in tutte le piattaforme) con path ai file che contengano degli spazi al loro interno. Quindi, affinchè i prototipi

Figura 4.17: Sflow in ambiente MacOS X

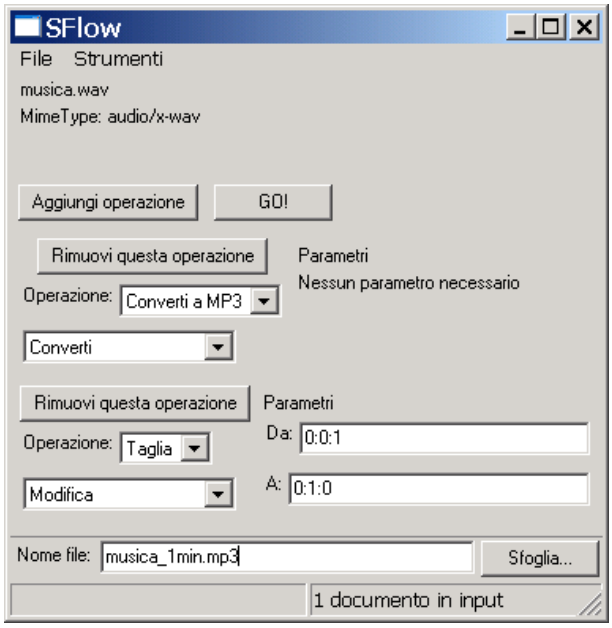

Figura 4.18: Sflow in ambiente Windows

funzionino correttamente, bisogna assicurarsi che i documenti che si intende trattare siano raggiungibili tramite dei percorsi che **non** contengano spazi al loro interno.

Il costrutto rdf:Seq

Al fine di permettere un'ordinamento nei dati RDF di una collezione è presente nella definizione dello standard RDF il costrutto rdf:Seq [10]. Tale costrutto permette appunto di definire sequenze ordinata di dati ed è di fondamentale importanza per l'implemetazione del modello descritto nel capitolo precedente. Viene infatti utilizzato nella definizione dei parametri di una operazione. Molto spesso infatti è necessario che i parametri siano presentati all'utente in maniera ordinata, si pensi ad esempio ad una operazione che effettua il rimpiazzo di una parola con un'altra in un file di testo. È prassi comune in questi casi avere a che fare con una maschera di input in cui viene prima specificata la parola da cercare e poi la parola con cui la si vuole sostituire. Bisogna quindi che i parametri da richiedere all'utente siano presentati in un certo ordine.

Redland, la libreria utilizzata per la manipolazione dei metadata, non permette ancora di recuperare dati dalle collezioni in maniera ordinata [34]. È stato quindi necessario specificare esplicitamente l'ordine di apparizione dei parametri all'interno della base semantica.

Di seguito è presentato uno stralcio di definizione di un'operazione. Come si può vedere è presente la proprietà `parameterOrder` che specifica l'ordine del relativo parametro

```
<rdf:Description rdf:nodeID="_sedTextReplace">
  <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
      mimetype#text_plain" />
  <apps:operation rdf:resource="http://sflow.org/data/2005-06/↩
      operation#textReplace" />
  <apps:parameters>
    <rdf:Seq>
      <rdf:li rdf:nodeID="_sedTextReplace_p1" />
      <rdf:li rdf:nodeID="_sedTextReplace_p2" />
    </rdf:Seq>
  </apps:parameters>
</rdf:Description>

<rdf:Description rdf:nodeID="_sedTextReplace_p1">
  <apps:parameterOrder>1</apps:parameterOrder>
  <apps:commandOption>-s</apps:commandOption>
  <apps:parameterLabel>Sostituisci</apps:parameterLabel>
  <apps:parameterType rdf:resource="&xsd;string" />
  <apps:parameterHelp>Parola da sostituire</apps:parameterHelp>
```

```
</rdf:Description>

<rdf:Description rdf:nodeID="_sedTextReplace_p2">
  <apps:parameterOrder>2</apps:parameterOrder>
  <apps:commandOption>-r</apps:commandOption>
  <apps:parameterLabel>con</apps:parameterLabel>
  <apps:parameterType rdf:resource="&xsd;string" />
  <apps:parameterHelp>Parola con cui sostituire</apps:parameterHelp>
</rdf:Description>
```

OPTIONAL e le Query

Molte delle proprietà definite nei vocabolari RDF presentati nel capitolo precedente sono opzionali, ossia non è obbligatoria la loro presenza al fine del funzionamento dell'applicazione. In questi casi la specifica del linguaggio di Query SPARQL prevede l'utilizzo dell'opzione `OPTIONAL` all'interno delle sue query (vedi sez. 2.1.3) al fine di recuperare eventuali dati opzionali.

Redland, la libreria utilizzata per la manipolazione dei metadata, non gestisce ancora correttamente tale parametro. Sembra infatti che quando, all'interno di una query, il soggetto di una tripla RDF è specificato, l'opzione `OPTIONAL` non funziona, o meglio, fa fallire la query sui dati opzionali qualora non esistano.

4.2 Application Helper

Il prototipo "Application Helper" implementa il "bonus" descritto nella sezione 3.5.1 del capitolo precedente. In sostanza, sfruttando la stessa base semantica utilizzata dall'applicazione *Sflow* è stata creata un'applicazione che fornisce una sorta di help sulle applicazioni installate nel sistema.

Fornendo come input il path di installazione di un'applicazione il programma presenta tramite una semplice interfaccia tutte le informazioni reperibili sull'applicazione in questione. Oltre ad informazioni accessorie quali la versione, il tipo di licenza, la pagina web del produttore e altro verrà presentata la lista delle operazioni fornite dall'applicazione. Scegliendo una delle operazioni proposte vengono presentate informazioni riguardo all'operazione scelta, quali una semplice spiegazione ed un'eventuale pagina HTML o XHTML che ne descrive le caratteristiche più in profondità (vedi fig. 4.19).

Figura 4.19: Schermata di AppHelp

4.2.1 Le classi del modello

Non sono state create nuove classi per definire il modello dei dati utilizzato da questa applicazione bensì si sono riutilizzate le classi descritte nella sezione 4.1.3.

4.2.2 Le classi dell'interfaccia utente

Di seguito vengono riportate, in forma tabellare, le classi, con la spiegazione dei relativi metodi, necessarie alla creazione dell'interfaccia utente dell'applicazione.

Tabella 4.21: Classe AppHelp

Classe `AppHelp`	
Si tratta della classe di partenza dell'applicazione. Ha il compito di creare il database semantico processando i file RDF disponibili nel sistema e di raccogliere i dati passati attraverso la linea di comando. È derivata dalla classe wxApp della libreria wxWidgets.	
Metodo	**Commento**
`OnInit(uri, model)`	Crea il database semantico e recupera i parametri passati a linea di comando (ossia il path all'eseguibile dell'applicazione da esaminare).

Tabella 4.22: Classe AppHelpMainFrame

Classe `AppHelpMainFrame`	
Crea e gestisce l'interfaccia utente. (La classe è derivata da wxFrame del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, ID, title, pathApp, model)`	Costruttore. Iniziallizza le variabili, raccoglie informazioni sui mimetype e prepara l'interfaccia grafica. In input ha bisogno di: **parent** la finestra padre (o NULL in questo caso); **ID** wxId della finestra che si andrà a creare; **title** Titolo della finestra che si andrà a creare; **pathApp** path completo all'applicazione da esaminare; **model** handle all'oggetto RDF.Model.
`RecoverApplication()`	Crea l'oggetto Application relativo all'applicazione presa in esame.
`RecoverOperations()`	Recupera dalla base semantica le operazioni implementate dall'applicazione e le eventuali relative categorie di appartenenza.
`FillTree()`	Riempie l'albero con le categorie e le operazioni. Le foglie rappresentano le operazioni.
`AddTreeCategory(parent, category)`	Metodo di appoggio per il metodo FillTree.
Continua nella pagina successiva	

Continua dalla pagina precedente	
`OnTreeSelected(event)`	Gestore dell'evento generato dalla selezione di una categoria (o di una operazione) nel relativo albero.
`OnAbout(event)`	Crea la finestra About.
`TimeToQuit(event)`	Chiusura dell'applicazione.

Tabella 4.23: Classe ApplicationPanel

Classe `ApplicationPanel`	
Presenta all'utente alcune operazioni riguardanti un'applicazione. (La classe è derivata da wxPanel del pacchetto wxWidgets).	
Metodo	**Commento**
`init(parent, ID, application)`	Costruttore. Il costruttore crea e riempie i controlli dell'interfaccia con dati relativi all'applicazione (nome, tipo di licenza, URL del prodotto, prezzo, path di installazione). In input ha bisogno della finestra padre, del wxID del pannello che si andrà a creare e dell'oggetto Application rappresentate l'applicazione.

Capitolo 5

Manuale utente

5.1 Cos'è Sflow

Quante volte capita di dover compiere una qualsiasi operazione su di un file e non saper bene quale programma utilizzare, se l'abbiamo installato sul nostro computer ed in che modo utilizzarlo al fine di raggiungere il nostro scopo? Il problema di fondo è che l'utente non dovrebbe preoccuparsi di quale applicativo usare ma solo dell'operazione che vuole effettuare.

Sflow si pone tra l'utente e gli applicativi "veri e propri" fornendo un'interfaccia che nasconde le altre applicazioni presentando all'utente una lista categorizzata di operazioni effettuabili dalle applicazioni installate nel sistema sui documenti passatigli come input. Le operazioni possono inoltre venir collegate tra di loro creando una sorta di *flusso di lavoro*. In figura 5.1 è illustrato un esempio di flusso a partire da una immagine.

Il concetto importante da capire è che Sflow, in base ai tipi di documento forniti in ingresso, presenterà all'utente **solamente le operazioni effettuabili su quei file dalle applicazioni installate nel sistema** e non una semplice lista di operazioni generiche o non correllate con i file in input. Di fatto, quindi, l'interfaccia di Sflow è dinamica e cambia in base ai documenti che si intende trattare.

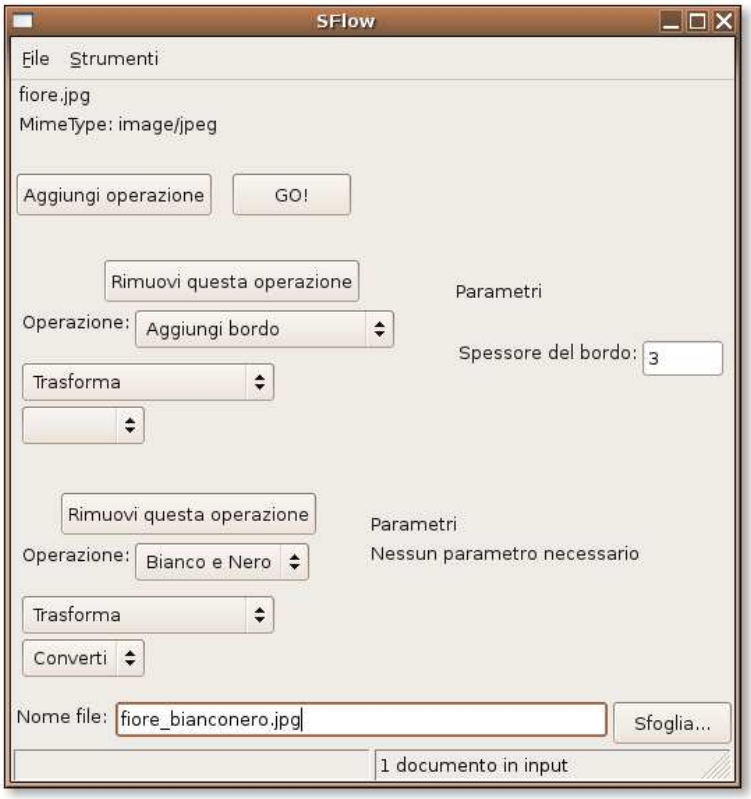

Figura 5.1: Esempio di flusso in Sflow

5.2 Cos'è AppHelp

AppHelp sfrutta la stessa *base di dati* utilizzata da Sflow ma fornisce un servizio diverso. A differenza di Sflow non accetta documenti in input bensì il path di qualche applicativo installato nel sistema e presenterà all'utente una lista categorizzata delle operazioni effettuabili dall'applicazione scelta. Si tratta quindi di una sorta di "help universale". Si veda la sezione 5.11 per un'immagine ed un esempio.

5.3 Installazione in ambiente Windows

Sia Sflow che AppHelp sono stati sviluppati utilizzando il linguaggio Python, è quindi necessario installare l'interprete Python nel proprio computer. Lo si può scaricare ed utilizzare gratuitamente dal sito ufficiale ossia http://www.python.org/download/.

Gli applicativi sviluppati necessitano inoltre l'installazione di alcune librerie esterne al fine di gestire i dati RDF e per creare l'interfaccia utente grafica:

librdf Redland (o librdf) può essere scaricata gratuitamente dal sito:
http://download.librdf.org/binaries/win32/1.0.2-2/
ed in particolare si deve scaricare ed installare il pacchetto
redland-python2.4-1.0.2.1-Win32-setup.exe

wxWidgets è la libreria base per il disegno dell'interfaccia utente. Anche questa può essere scaricata gratuitamente dal sito:
http://prdownloads.sourceforge.net/wxwindows/wxMSW-2.6.2-Setup.exe

wxPython si tratta della libreria di collegamento tra il linguaggio di programmazione Python e la libreria wxWidgets. Ancora una volta è possibile scaricarla gratuitamente dal sito:
http://prdownloads.sourceforge.net/wxpython/wxPython2.6-win32-unicode-2.6.1.0-py24.exe

Una volta installate le librerie precedenti è possibile installare i programmi Sflow e AppHelp.

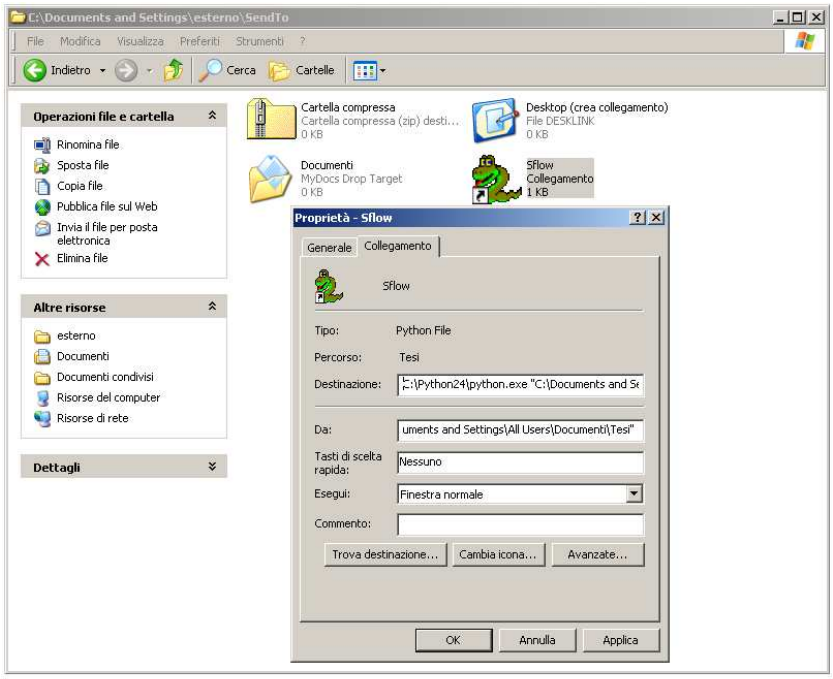

Figura 5.2: Creazione link per Windows

5.3.1 Integrazione con Windows

Al fine di rendere più agevole l'utilizzo di Sflow e AppHelp è possibile farli apparire come voci del menu contestuale "Invia a...". A tale scopo bisogna creare due link, uno al file Flusso.py per Sflow ed uno al file AppHelp.py per l'applicazione AppHelp. I link vanno posti nella cartella "C:\Document and Settings\Utente\Send To" dove "Utente" è il nome dell'utente su quella macchina (vedi fig. 5.2).

A questo punto cliccando con il tasto destro del mouse su uno o più file è possibile lanciare Sflow (o AppHelp) su di essi come illustrato in figura 5.3.

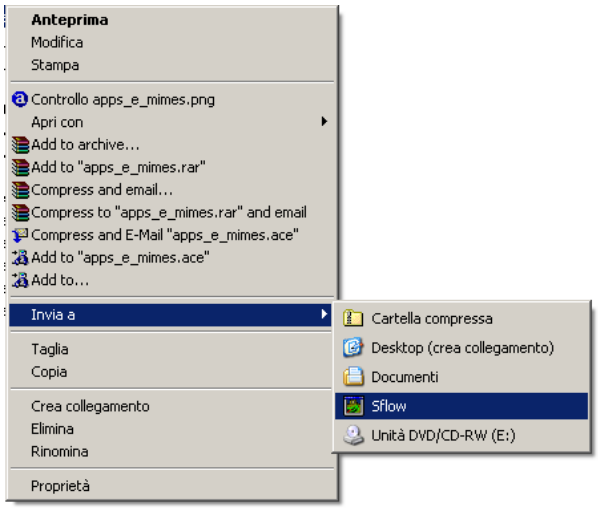

Figura 5.3: Sflow integrato con Windows

5.4 Installazione in ambiente MacOS X

Sflow e AppHelp sono stati testati con le versioni 10.3 e 10.4 del sistema operativo MacOS X. L'installazione è diversa a seconda della versione del sistema operativo. In entrambi i casi, comunque, va dapprima compilata la libreria librdf ed il suo binding con il linguaggio di programmazione Python (già presente nel sistema operativo). Per poterle compilare è necessario prima installare i "Developer Tools" presenti nel DVD di installazione del sistema operativo (o su di un CD a se stante). Installati i Developer Tools si può proseguire con la compilazione della libreria `librdf`:

- Scaricare i file redland-1.0.2.tar.gz e redland-bindings-1.0.2.1.tar.gz da http://download.librdf.org/source/
- Decomprimere il file redland-1.0.2.tar.gz dando, da terminale, il comando `tar xfz redland-1.0.2.tar.gz`;
- Entrare nella directory con `cd redland-1.0.2`;
- Compilare ed installare la libreria con i comandi `./configure ; make ; sudo make install` (sono richiesti i privilegi di amministratore).

- Tornare nella directory dove sono stati scaricati i file e decomprimere il file redland-bindings-1.0.2.1.tar.gz con il comando: `tar xfz redland-bindings-1.0.2.1.tar.gz`;
- Entrare nella nuova directory con `cd redland-bindings-1.0.2.1`;
- Compilare ed installare la libreria con i comandi `./configure --with-python ; make ; sudo make install` (sono richiesti i privilegi di amministratore).

Di seguito verrà descritta passo passo il proseguio dell'installazione in base alla versione del sistema operativo disponibile.

5.4.1 MacOS X 10.3 (Panther)

Ci si deve dotare della libreria wxWidgets e di wxPython ossia il suo collegamento con il linguaggio di programmazione Python:

wxWidgets Vanno scaricati i sorgenti dal sito:
http://prdownloads.sourceforge.net/wxwindows/wxMac-2.6.2.tar.gz
decompressi e installati seguendo quanto scritto nel file `install-mac-2.6.2.txt`

wxPython La si può trovare al link:
http://prdownloads.sourceforge.net/wxpython/wxPython2.6-osx-unicode-2.6.1.0-macosx10.3-py2.3.dmg

A questo punto è possibile scaricare le applicazioni Sflow e AppHelp.

5.4.2 MacOS X 10.4 (Tiger)

Per la versione 10.4 di MacOS X non occorre fare altro se non scaricare le applicazioni Sflow e AppHelp.

5.5 Installazione in ambiente GNU/Linux

Al fine di poter utilizzare Sflow ed AppHelp è necessario che nel sistema siano presenti:

- Il linguaggio di programmazione Python almeno nella versione 2.3 scaricabile da:
  http://www.python.org/download
- La libreria grafica wxWidgets almeno nella versione 2.5:
  http://www.wxwidgets.org
- Il binding tra Python e wxWidgets:
  http://www.wxpython.org
- La libreria Redland almeno nella versione 1.0 con relativo binding per Python:
  http://www.librdf.org

I sistemi GNU/Linux sono eterogenei è quindi difficile indicare le modalità di installazione per i precedenti pacchetti.

Fino ad ora Sflow ed AppHelp sono stati testati con distribuzioni Debian (release testing) e Ubuntu (versione Hoary). In questi casi i pacchetti da installare tramite il comando `apt-get install` sono:

- `python2.3` o `python2.4`
- `libwxgtk2.5.3`
- `wxpython2.5.3`
- `librdf0`
- `python2.3-rdflib` o `python2.4-rdflib`

5.5.1 Integrazione con Gnome

Il desktop environment open-source Gnome [28] viene fornito con un file manager di nome Nautilus [31]. Risulta piuttosto semplice l'integrazione di Sflow in quest'ambito. Nautilus, infatti, prevede di poter richiamare degli script partendo da un menu contestuale associato ai documenti. Allo script vengono passati i nomi (completi di path) dei file selezionati dall'utente.

Volendo quindi integrare Sflow con Nautilus è sufficiente creare un semplice script di shell da porre poi nella directory `<home>/.gnome2/nautilus-scripts`:

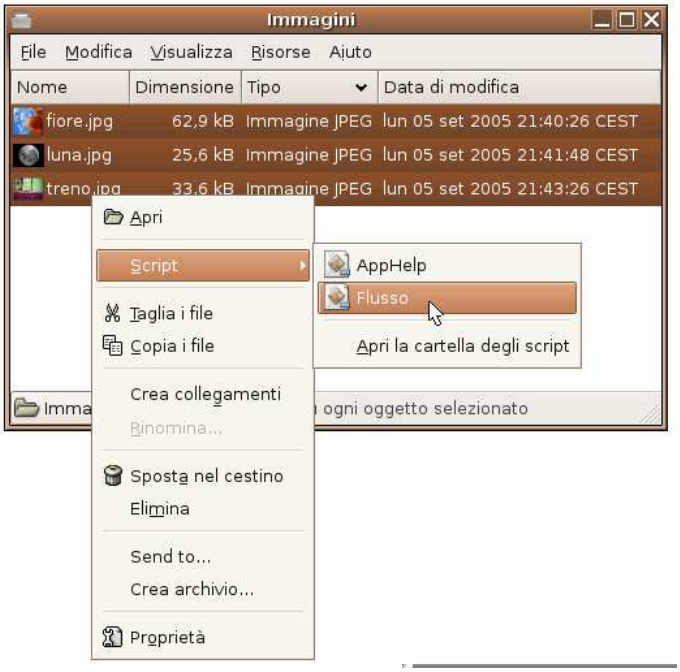

Figura 5.4: Sflow integrato con Nautilus

```
#!/bin/bash
#Title=SFlow

python <path di installazione di Sflow>/Flusso.py $↩
    NAUTILUS_SCRIPT_SELECTED_FILE_PATHS
```

A questo punto selezionando uno o più file da una finestra di Nautilus e premendo il tasto destro del mouse si otterrà un menu contestuale da cui poter accedere all'applicazione Sflow partendo dal sottomenu "Scripts" (vedi figura 5.4).

5.6 Il repository dei file RDF

Le funzionalità di Sflow e AppHelp sono strettamente legate a quanto viene descritto nei file RDF a corredo degli applicativi esterni. A tal fine è necessario creare una directory su cui andare a posizionare tali documenti. La directory può trovarsi in qualsiasi posizione del file-system e può avere un nome qual-

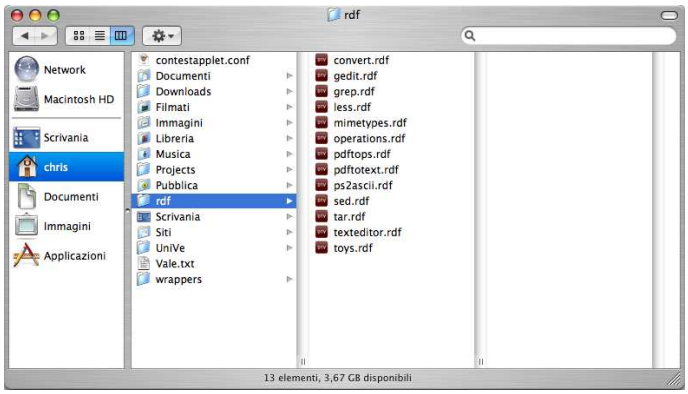

Figura 5.5: Esempio di posizionamento della cartella contenente i documenti RDF

siasi (vedi figura 5.5). Tale directory andrà indicata a Sflow (e AppHelp) nel momento della loro prima esecuzione e sarà possibile modificarla in seguito dalla voce "Preferenze" del menu "File".

5.7 Gli applicativi esterni

Sia Sflow che AppHelp si appoggiano su applicazioni esterne per poter compiere le proprie funzionalità. Tali applicativi "informano" Sflow (o AppHelp) delle relative capacità attraverso i documenti RDF di cui si parlava nella sezione precedente. È quindi necessario che qualora un'applicazione voglia integrarsi con Sflow debba fornire un documento redatto secondo una certa sintassi e vocabolario in cui descrive le proprie caratteristiche e funzionalità.

Prendiamo ad esempio l'applicazione `convert` della suite ImageMagick [30]. Questo applicativo a linea di comando permette di eseguire svariate operazioni su dei file immagine e, caratteristica per cui è stato scelto come esempio, è disponibile gratuitamente (ed in più è open-source) per molte piattaforme (GNU/Linux, MacOS X e Windows compresi).

Per prima cosa va notato come l'ordine e la sintassi con cui `convert` si aspetta i parametri non sono gli stessi previsti da Sflow. `convert` infatti si aspetta dapprima il/i file in input, seguiti dall'operazione segnalata da un trattino seguito dall'identificativo dell'operazione e per finire il nome del file in output. Sflow, invece, richiama gli applicativi esterni passando come primo

parametro il/i file in input (e qui concorda con `convert`) seguito dall'operazione che può essere specificata o con un trattino seguito da una sola lettera (non concorda quindi con `convert`) o da due trattini seguiti dall'identificativo dell'operazione, finisce poi specificando i file in output introducendoli con `-o` o `--output`.
Ad esempio il comando in uscita da Sflow:
`convert input.jpg --rotate 90 -o output.jpg`
andrebbe convertito in:
`convert input.jpg -rotate 90 output.jpg`

È quindi necessaria un'operazione di ordinamento che può essere eseguita utilizzando un semplice script come ad esempio il seguente scritto in linguaggio Python:

```
#!/usr/bin/python
import os
import sys
from optparse import OptionParser

class convert_wrp:
    def __init__(self):
        print "Convert Wrapper"
        self.parser = OptionParser()

        self.options_list = ["--rotate", "--flop", "--monochrome"]

        self.parser.add_option("--rotate", action="store",
                          type="string", dest="rotate")
        self.parser.add_option("--flop", action="store_true",
                          dest="flop")
        self.parser.add_option("--monochrome", action="store_true",
                          dest="monochrome")

        self.parser.add_option("-o", "--output", action="append",
                          type="string", dest="outputFile")

        (self.options, self.args) = self.parser.parse_args()


    def performOperation(self):
        command = "/usr/bin/convert"

        lParams = []
        lParams.append(command)

        for inFile in self.args:
            lParams.append(inFile)

        if self.options.rotate != None:
            lParams.append("-rotate")
```

```
            lParams.append(self.options.rotate)
        if self.options.flop:
            lParams.append("-flop")
        if self.options.monochrome:
            lParams.append("-monochrome")

        for outFile in self.options.outputFile:
            lParams.append(outFile)

        result = os.spawnv(os.P_WAIT, command, lParams)
        return result


convert = convert_wrp()
sys.exit(convert.performOperation())
```

A questo punto è possibile redigere il documento RDF relativo a `convert` al fine di rendere note le proprie capacità di poter effettuare le operazioni di conversione in bianco e nero, ribaltamento orizzontale e rotazione sulle immagini jpeg e png. Ecco come (le linee sono numerate ed il codice spezzato per poterlo commentare più agevolmente):

```
 1. <?xml version="1.0"?>
 2. <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#↩
     ">]>
 3. <rdf:RDF
 4.  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 5.  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 6.  xmlns:mm="http://sflow.org/data/2005-06/mimetype#"
 7.  xmlns:apps="http://sflow.org/schema/2005-06/applications#"
 8.  xmlns:ops="http://sflow.org/schema/2005-06/operations#"
 9.  xml:base="http://www.imagemagick.org/applications">
10.
```

Le righe da 1 a 9 sono l'intestazione del documento. Ne dichiarano il tipo ed i namespace utilizzati. Da notare la linea 9 in cui agli applicativi della suite ImageMagick viene dato un namespace fittizio a cui faranno riferimento le applicazioni e le operazioni descritte.

```
11.
12. <ops:Operation rdf:ID="imageRotate">
13.   <ops:name>Ruota</ops:name>
14.   <ops:returnMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#sameasinput" />
15.   <ops:shortComment>Ruota, in senso orario, l'immagine di un certo↩
     numero di gradi</ops:shortComment>
16. </ops:Operation>
17.
18. <ops:Operation rdf:ID="imageFlipH">
19.   <ops:name>Ribalta orizzontalmente</ops:name>
```

```
20.   <ops:returnMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#sameasinput" />
21.   <ops:shortComment>Ribalta l'immagine orizzontalmente</↩
    ops:shortComment>
22. </ops:Operation>
23.
24. <ops:Operation rdf:ID="imageBlackWhite">
25.   <ops:name>Bianco e Nero</ops:name>
26.   <ops:returnMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#sameasinput" />
27.   <ops:shortComment>Converte l'immagine in bianco e nero</↩
    ops:shortComment>
28.   <ops:hasCategory rdf:resource="#ImageConvert" />
29. </ops:Operation>
30.
```

Le righe da 12 a 29 descrivono le operazioni che vogliamo trattare tra quelle che `convert` sa compiere.

```
31.
32. <apps:Application rdf:ID="convert">
33.   <apps:name>Convert</apps:name>
34.   <apps:installationPath>/usr/wrappers/</apps:installationPath>
35.   <apps:execCommand>convert_wrp.py</apps:execCommand>
36.   <apps:version>6.0</apps:version>
37.   <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#image_jpeg" />
38.   <apps:canWrite rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#image_png" />
39.   <apps:canPerform>
40.     <rdf:Bag>
41.       <rdf:li rdf:nodeID="_convertImageRotate" />
42.       <rdf:li rdf:nodeID="_convertImageFlipH" />
43.       <rdf:li rdf:nodeID="_convertImageBlackWhite" />
44.     </rdf:Bag>
45.   </apps:canPerform>
46. </apps:Application>
47.
48. <rdf:Description rdf:nodeID="_convertImageRotate">
49.   <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#image_jpeg" />
50.   <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#image_png" />
51.   <apps:operation rdf:resource="#imageRotate" />
52.   <apps:parameters>
53.     <rdf:Seq>
54.       <rdf:li rdf:nodeID="_convertImageRotate_p1" />
55.     </rdf:Seq>
56.   </apps:parameters>
57. </rdf:Description>
58.
59. <rdf:Description rdf:nodeID="_convertImageRotate_p1">
60.   <apps:commandOption>--rotate</apps:commandOption>
```

```
61.   <apps:parameterLabel>Angolo della rotazione:</↩
    apps:parameterLabel>
62.   <apps:parameterType rdf:resource="&xsd;decimal" />
63.   <apps:parameterHelp>Angolo in gradi centigradi della rotazione ↩
    in senso orario</apps:parameterHelp>
64. </rdf:Description>
65.
```

Le righe da 32 a 46 descrivono l'applicativo `convert` (path di installazione, versione, tipi di documenti che sa gestire e operazioni che sa effettuare).

Le righe da 48 a 64 descrivono l'operazione di rotazione di un'immagine. Viene specificato a quali tipi di immagine può essere effettuata e i parametri che necessita per compiere l'operazione.

```
66.
67. <rdf:Description rdf:nodeID="_convertImageFlipH">
68.   <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#image_jpeg" />
69.   <apps:operation rdf:resource="http://sflow.org/data/2005-06/↩
    operation#imageFlipH" />
70.   <apps:parameters>
71.     <rdf:Seq>
72.       <rdf:li rdf:nodeID="_convertImageFlipH_p1" />
73.     </rdf:Seq>
74.   </apps:parameters>
75. </rdf:Description>
76.
77. <rdf:Description rdf:nodeID="_convertImageFlipH_p1">
78.   <apps:commandOption>--flop</apps:commandOption>
79. </rdf:Description>
80.
81. <rdf:Description rdf:nodeID="_convertImageBlackWhite">
82.   <apps:toMimeType rdf:resource="http://sflow.org/data/2005-06/↩
    mimetype#image_jpeg" />
83.   <apps:operation rdf:resource="http://sflow.org/data/2005-06/↩
    operation#imageBlackWhite" />
84.   <apps:parameters>
85.     <rdf:Seq>
86.       <rdf:li rdf:nodeID="_convertImageBlackWhite_p1" />
87.     </rdf:Seq>
88.   </apps:parameters>
89. </rdf:Description>
90.
91. <rdf:Description rdf:nodeID="_convertImageBlackWhite_p1">
92.   <apps:commandOption>--monochrome</apps:commandOption>
93. </rdf:Description>
94. </rdf:RDF>
```

Nelle altre righe (da 67 a 94) vengono descritte le altre operazioni e chiuso il documento RDF.

5.7.1 L'utilità di Py2Exe

Nella sezione precedente si è visto come sia stato necessario creare uno script di "adattamento" per convertire la modalità di passaggio dei parametri all'applicativo esterno. Volendo utilizzare quello script in ambiente Windows è conveniente convertirlo in file eseguibile attraverso l'utilizzo dell'utilità Py2Exe scaricabile gratuitamente dal sito:
http://starship.python.net/crew/theller/py2exe/
Per poter creare l'eseguibile è necessario creare il file `setup.py`:

```
from distutils.core import setup
import py2exe

setup(
    options = {"py2exe": {"packages": ["encodings"]}},
    console = ["convert_wrp.py"])
```

e richiamare, da terminale, il comando:
`python setup.py py2exe`
in questo modo verrà creata una sottodirectory `dist` in cui si troverà l'eseguibile desiderato.

5.8 Invocazione di Sflow

Nel caso in cui non sia possibile integrare Sflow nel gestore di file del sistema operativo (come invece è possibile fare in ambiente Gnome, vedi punto 5.5.1, o in Windows, vedi 5.3.1) è sempre possibile invocarli tramite linea di comando.

In particolare è comodo innanzitutto assicurarsi che sia presente nella variabile di ambiente `PATH` la directory in cui è installato il comando python o pythonw.

Volendo quindi, ad esempio, lanciare Sflow per poter elaborare il file `prova.jpg` che si trova nella directory `/home/pippo/` è sufficiente lanciare il comando:
`python <dir>/Flusso.py /home/pippo/prova.jpg`
dove `<dir>` è la directory in cui è stato installato Sflow.

ATTENZIONE! In ambiente MacOS X va utilizzato `pythonw` al posto di `python`.

Figura 5.6: Sflow: pannello iniziale

5.9 Utilizzo di Sflow

Una volta specificata la posizione da cui attingere le informazioni (vedi punto 5.6) Sflow si presenterà sotto forma di una semplice finestra in cui viene riassunto il nome del file che su cui si sta lavorando, il mime-type del file, una lista di applicazioni in grado di gestire il file in lettura o scrittura, un pulsante "Apri con" (vedi figura 5.6) ed un pulsante "Aggiungi Operazione".

Il pulsante "Apri con" ha la funzione di aprire il file di partenza con l'applicazione scelta nella relativa lista.

Il pulsante "Aggiungi Operazione" aggiunge alla finestra un nuovo pannello (vedi figura 5.7) da cui poter scegliere quali operazioni compiere sul file in input ad Sflow. Da notare che le operazioni non vengono eseguite da Sflow bensì da altri applicativi installati nel sistema che rendono note le proprie capacità attraverso i documenti RDF di cui si parla nel punto 5.6.

Le operazioni possono essere categorizzate in base a categorie gerarchiche la cui visualizzazione è demandata ad una serie di combo-box (vedi figura 5.8). Scegliendo una categoria qualora questa abbia delle sottocategorie verranno visualizzate in un nuovo combo-box che apparirà sotto al combo della categoria scelta. Ovviamente ogni volta che si sceglie una categoria il combo-box contenente la lista delle possibili operazioni cambierà contenuto.

Una volta scelta un'operazione potrebbe essere necessario dover specificare qualche parametro al fine di poterla poi eseguire (vedi figura 5.9).

Qualora l'operazione scelta producesse uno o più file in output sarebbe necessario fornirne un nome e a tal scopo apparirebbe un nuovo pannello,

Figura 5.7: Sflow: pannello operazione

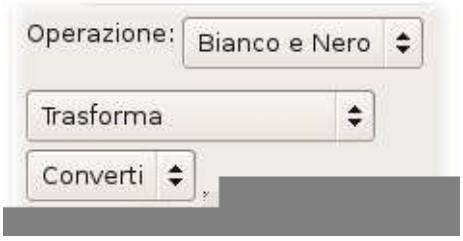

Figura 5.8: Sflow: Particolare sulla lista delle categorie

Figura 5.9: Sflow: Operazione con parametri

Figura 5.10: Sflow: Particolare sul pannello dei nomi in output

come illustrato in figura 5.10.

Da notare che il pulsante "Aggiungi Operazione" è ancora presente e la sua pressione consente di aggiungere via via nuove operazioni in cascata permettendo la creazione di un vero e proprio flusso di lavoro (vedi figura 5.11) che può anche venire convenientemente salvato (e ricaricato) tramite le apposite voci del menu "File".

Una volta definito completamente il flusso di operazioni è possibile "eseguirlo" premendo il pulsante "GO!". Verranno quindi eseguite le varie operazioni prsenti, una di seguito all'altra e lo stato complessivo di avanzamento del lavoro sarà visualizzato da un'apposita progress-bar (vedi figura 5.12).

Figura 5.11: Sflow: Esempio di flusso di lavoro

Figura 5.12: Sflow: Particolare della progress-bar

Figura 5.13: Sflow: complessità delle applicazioni

5.9.1 Scelta della complessità delle applicazioni

Gli applicativi di cui Sflow andrà a sfruttare le potenzialità possono essere di complessità diversa. In particolare sono categorizzati utilizzando una scala da 1 a 5. Il programma Sflow gestisce queste informazioni attraverso una slide-bar (vedi figura 5.13) che rende possibile per l'utente scegliere il grado di difficoltà desiderato.

Normalmente la slide-bar è nascosta ed il livello di complessità è settato ad un valore medio pari a 2. Qualora l'utente volesse cambiarlo può agire sulla voce "Mostra complessità" del menu "Strumenti".

5.9.2 Applicativi multipli

Può capitare che più appplicazioni siano in grado di eseguire la medesima operazione. In questi casi il programma Sflow si comporta scegliendo autonomamente una tra le possibili applicazioni in grado di compiere l'operazione in questione. Resta comunque possibile per l'utente poter scegliere esplicitamente a quale applicazione far compiere l'operazione. È infatti sufficiente agire sulla voce "Mostra applicazioni" del menu "Strumenti" per far apparire dei combo-box nei vari *pannelli operazione* da cui poter scegliere l'applicativo preferito (vedi figura 5.14).

Figura 5.14: Sflow: applicativi multipli

5.10 Invocazione di AppHelp

Nel caso in cui non sia possibile integrare AppHelp nel gestore di file del sistema operativo (come invece è possibile fare in ambiente Gnome, vedi punto 5.5.1) è sempre possibile invocarlo tramite linea di comando.

In particolare è comodo innanzitutto assicurarsi che sia presente nella variabile di ambiente `PATH` la directory in cui è installato il comando python o pythonw.

Volendo quindi, ad esempio, lanciare AppHelp per poter avere informazioni sull'applicativo `convert` installato nella directory `/usr/bin/` è sufficiente lanciare il comando:
`python <dir>/AppHelp.py /usr/bin/convert`
dove `<dir>` è la directory in cui è stato installato AppHelp.

ATTENZIONE! In ambiente MacOS X va utilizzato `pythonw` al posto di `python`.

5.11 Utilizzo di AppHelp

AppHelp è sostanzialmente un'applicativo di consultazione. Non fornisce funzionalità bensì un help riguardante altri applicativi installati nel sistema. Si presenta con un'interfaccia molto semplice. Una parte superiore in cui sono presentate informazioni riguardanti l'applicativo scelto (nome, versione, pagi-

Figura 5.15: Interfaccia di AppHelp

na web, tipo di licenza, prezzo etc.) ed un'albero nella parte sinistra. Le foglie dell'albero rappresentano le singole operazioni effettuabili dall'applicativo che si sta esaminando mentre i nodi interni sonno le categorie secondo cui sono organizzate le operazioni (vedi figura 5.15).

Selezionando un'operazione verranno visualizzate nella parte destra dell'interfaccia tutte le informazioni che la riguardano.

Capitolo 6

Conclusioni

6.1 Risultati ottenuti

Concludendo, la presente tesi ha presentato un modello per la rappresentazione e dichiarazione di metadati associati a tipi di documento e applicazioni al fine di rendere più agevole il rapporto tra utente comune e macchina (vedi capitolo 3). Il concetto fondamentale era la volontà di nascondere all'utente l'entità *applicazione* rendendo invece visibili le singole *funzionalità o operazioni* da essa offerte. Da qui la possibilità di definire dei *flussi di lavoro* a partire da uno o più documenti.

È inoltre stata dimostrata la fattibilità del progetto con l'implementazione di due prototipi di applicazioni in grado di sfruttare la base semantica creata (vedi capitolo 4).

6.2 Possibili approfondimenti

6.2.1 Cercare l'applicazione corretta

Nella sezione 3.2.1 è descritta la classe OWL Mimetype. Tra le proprietà disponibili è presente `canApply` che lega un mime-type con le possibili operazioni applicabili su di esso. In realtà questa proprietà non è utilizzata dal programma Sflow in quanto per risalire alle operazioni effettuabili sui documenti scelti dall'utente si parte dalla classe Application che a sua volta dichiara le operazioni in grado di eseguire.

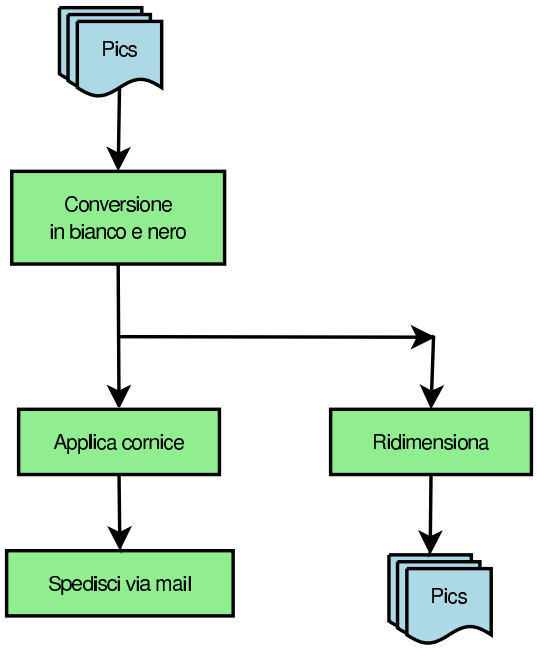

Figura 6.1: Possibile flusso di operazioni multiplo

La proprietà `canApply` può quindi essere utilizzata in altro modo. Si può immaginare di trovarsi nella situazione in cui le applicazioni installate nel sistema non riescano a fornire l'operazione desiderata dall'utente. In questo caso può risultare utile poter interrogare una base di dati, magari remota, per poter effettivamente capire se l'operazione desiderata sia effettivamente possibile sul documento scelto e quali applicazioni, anche non installate nel sistema, siano in grado di eseguirla. La query può inoltre venir raffinata potendo scegliere la licenza preferita (open-source, freeware, shareware, commerciale) e magari ponendo dei limiti sul prezzo (grazie alle proprietà della classe Application, vedi sez. 3.3.1).

6.2.2 Suddivisione del flusso

A volte potrebbe risultare utile poter suddividere il flusso in più rami di elaborazione. Si prenda ad esempio il flusso in figura 6.1.

6.2.3 Iniziare il flusso da interrogazioni su metadata

Negli ultmi tempi stanno aumentando le implementazioni di filesystem aventi supporto per metadata (si pensi alla tecnologia Spotlight [12] in ambiente MacOS, Beagle [25] in ambiente GNU/Linux o WinFS [11] sotto Windows). Poter iniziare il flusso di operazioni da una query su questo tipo di filesystem aiuterebbe l'utente a superare l'altro grosso problema per gli utenti inesperti, ossia il concetto stesso di filesystem gerarchico.

Allo stesso modo poter partire da query effettuate su database aventi supporto per metadata potrebbe essere molto utile così come da interrogazioni a motori di ricerca su Internet.

In sostanza grande enfasi andrebbe posta nel nascondere il filesystem all'utente inesperto. Le entità importanti dovrebbero essere solamente i documenti e le operazioni effettuabili su di essi a prescindere dalla loro locazione e dalle applicazioni.

6.2.4 Integrazione con applicazioni remote

Sflow può essere visto come un *creatore dinamico di interfaccia utente* per applicativi nati per essere eseguiti attraverso la linea di comando. Attualmente lavora con le applicazioni installate *in locale*, ossia nello stesso computer in cui è presente Sflow, ma nulla vieterebbe di studiare qualche tipo di protocollo per poterlo mettere in comunicazione con applicazioni remote, ossia su altri computer nella stessa rete locale o anche geografica (internet).
Potrebbe, ad esempio, essere integrato con XMLRPC [13] così da permettere agli utenti di utilizzare applicativi remoti proprio come se fossero in locale e di creare *flussi di lavoro misti* ossia composti di operazioni svolte in locale e operazioni svolte in remoto con assoluta trasparenza.

6.2.5 Andare oltre il mimetype

L'idea di base del modello descritto nel capitolo 3 è quella di creare una base semantica in cui i mimetype dei documenti presenti in un computer vengano associati alle applicazioni (presenti nel sistema) in grado di gestirli e, soprattutto, con le operazioni in grado di essere applicate ai documenti in questione. Si potrebbe però ragionare ad un *livello più fine* ossia considerare tipi di dato più semplici di un documento quali ad esempio tipi di dato elementari come numeri interi, stringhe, numeri decimali o strutture più complesse composte

da tipi elementari ma ben definite attraverso l'utilizzo di RDF e OWL.
Si pensi ad esempio di avere a disposizione, tramite una query su di un database, due collezioni distinte di oggetti quali i valori della temperatura atmosferica e una struttura contenente i valori della longitudine e latitudine delle città a cui si riferiscono le temperature. Se nel nostro sistema fosse presente un'applicazione che dati in ingresso dei valori numerici e dei punti geografici disegnasse una cartina riportando i valori suddetti ecco che un Sflow "potenziato" potrebbe riconoscerla ed utilizzarla.
In qualche modo, quindi, Sflow diventerebbe un *agente software intelligente* di cui si parlava nel capitolo 2 nella sezione Semantic Web (2.1).

Ringraziamenti

Lo svolgimento di questa tesi ha richiesto più di un anno di tempo di lavoro. In questo periodo varie persone, oltre al sottoscritto, ne hanno contribuito alla realizzazione sia direttamente che indirettamente.
Innanzitutto devo ringraziare il professor Massimo Marchiori per aver creduto nella mia idea e nell'avermi aiutato, sin dall'inizio, a delinearne meglio i contorni e a definire degli obiettivi concreti.
Devo poi ringraziare i miei familiari tutti, amici e conoscenti per aver sopportato i miei cambi di umore (non pochi...) legati all'andamento della tesi durante questo periodo.
Ultimi, ma non ultimi, devo ringraziare i miei genitori, Renzo e Firenza, per avermi sostenuto (finanziariamente e non) in tutti questi anni da "studente". Grazie infinite.
Un traguardo è ormai vicino ma la corsa non è affatto finita.

Appendice A

Glossario

ASCII È l'acronimo di American Standard Code for Information Interchange (ovvero Codice Standard Americano per lo Scambio di Informazioni).
Si tratta di un sistema di codifica dei caratteri a 7 bit comunemente utilizzato nei calcolatori, proposto dall'ingegnere dell'IBM Bob Bemer nel 1961, e successivamente accettato come standard.
Alla specifica iniziale basata su codici di 7 bit fecero seguito negli anni molte proposte di estensione ad 8 bit, con lo scopo di raddoppiare il numero di caratteri rappresentabili. Nei PC IBM si fa per l'appunto uso di una di queste estensioni, ormai standard di fatto, chiamata extended ASCII o high ASCII. Nell'extended ASCII questi ulteriori simboli sono vocali accentate, simboli semigrafici e altri simboli di uso meno comune.

Classe In informatica, il termine classe può indicare, a seconda del contesto, una categoria di oggetti, un tipo di dati, o la implementazione di un tipo di dati. Queste tre accezioni si trovano rispettivamente (soprattutto) nella analisi orientata agli oggetti, nella progettazione orientata agli oggetti e nei linguaggi di programmazione orientati agli oggetti.
Sostanzialmente una classe descrive un tipo di oggetti (una categoria di entità) in termini di un insieme di variabili interne o variabili d'istanza di cui tali oggetti sono dotati (attributi) e un insieme di procedure dette metodi che possono essere eseguite su di essi (operazioni).

Cross Platform Con cross platform si intende un linguaggio di programmazione, un software o un dispositivo hardware in grado di funzionare su più di una piattaforma diversa (Microsoft Windows, UNIX, Macintosh,

etc.).
La maggior parte dei linguaggi di programmazione sono in un certo senso cross-platform in quanto altro non sono che linguaggi comprensibile dall'uomo per istruire un processore ad eseguire determinate operazioni e quindi non c'è dipendenza con il particolare sistema operativo. Comunque per eseguire compiti quali creare l'interfaccia utente di un applicativo o accedere alle varie perifieriche della macchina bisognerà far ricorso a varie librerie che sono generalmente legate al particolare ambiente operativo.
Sono quindi nati linguaggi disegnati per girare su *virtual machine*, come ad esempio Java, e bypassare quindi le problematiche legate al sistema operativo. Ma esistono anche librerie cross-platform, come ad esempio wxWidgets [39] nate per favorire lo sviluppo di applicativi multipiattaforma senza dover ricorrere a linguaggi dotati di virtual machine.

Metadati Letteralmente *dato circa un (altro) dato*, è l'informazione che descrive un altro insieme di dati.
Un esempio tipico di metadati è costituito dalla scheda del catalogo di una biblioteca, la quale contiene informazioni circa il contenuto e la posizione di un libro: questi sono dati riguardanti i dati nel libro cui si fa riferimento mediante la scheda.

Mime-type Il Multipurpose Internet Mail Extensions, o più brevemente MIME è un protocollo Internet che estende l'SMTP (Simple Mail Transfer Protocol) per permettere ai dati non codificati secondo il classico standard ASCII a 7 bit, come dati video, suoni e file binari, di essere trasmessi tramite la posta elettronica senza dover prima essere convertiti in formato ASCII; questa operazione viene compiuta mediante l'uso di vari tipi di MIME, che descrivono il contenuto di un documento. Un'applicazione compatibile MIME che invia un file, assegna un tipo di MIME al file. L'applicazione ricevente, che deve essere anch'essa compatibile MIME, fa riferimento a un elenco standard di documenti organizzati per tipi e sottotipi di MIME al fine di interpretare il contenuto del file. Per esempio, un tipo di MIME è text che ha un certo numero di sottotipi, tra i quali plain e html.
Allo stesso modo i documenti presenti in memoria sui normali computer desktop sono classificati secondo il loro MIME o altresì MimeType.
L'organismo internazionale che ha il compito di assegnare i mimety-

pe ai diversti tipi di documento è IANA (Internet Assigned Numbers Authority) [29].

Oggetto Un oggetto è una istanza di una classe (a volte di più di una). Un oggetto occupa memoria, la sua classe definisce come sono organizzati i dati in questa memoria.
Ogni oggetto possiede tutti gli attributi definiti nella classe, ed essi hanno un valore, che può mutare durante l'esecuzione del programma come quello di qualsiasi variabile. Sintatticamente, i metodi di una classe vengono invocati su un particolare oggetto, e ricevono come parametro implicito l'oggetto su cui sono stati invocati.

Ontologia Nell' informatica, una ontologia è il tentativo di formulare uno schema concettuale esaustivo e rigoroso nell'ambito di un dato dominio; si tratta generalmente di una struttura dati gerarchica che contiene tutte le entità rilevanti, le relazioni esistenti fra di esse, le regole, gli assiomi, ed i vincoli specifici del dominio. L'uso del termine ontologia nell'informatica è derivato dal precedente uso dello stesso termine in filosofia, dove ha il significato dello studio dell'essere o dell'esistere, così come le fondamentali categorie e delle relazioni tra esse.

Pipeline Il termine pipeline in informatica e in elettronica si riferisce a un manufatto composto da più elementi. Ogni elemento provvede a ricevere in ingresso un dato o un segnale, ad elaborarlo e poi a trasmetterlo all'elemento successivo. Quindi il flusso di dato o di segnali percorre tutti gli elementi fino all'ultimo elemento come quando una conduttura è attraversato da un fluido infatti il termine pipeline in inglese indica una tubatura o una conduttura.

Semantica Il termine semantica (dal greco semantikos, significato, derivato da sema, segno) è usato in molte diverse accezioni, tutte attinenti al concetto di *significato* di un messaggio in senso ampio. In alcuni casi, esso viene contrapposto a sintassi, intesa, in senso ampio, come *forma esteriore* del messaggio.

SMTP Simple Mail Transfer Protocol (SMTP) è il protocollo standard per la trasmissione via internet di e-mail. In italiano si potrebbe tradurre come Protocollo elementare di trasferimento postale.

È un protocollo relativamente semplice, testuale, nel quale vengono specificati uno o più destinatari di un messaggio, verificata la loro esistenza, e infine il messaggio viene trasferito.

UML In ingegneria del software, UML (Unified Modeling Language, linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica standard basato su concetti legati ai linguaggi ad oggetti.
Il nucleo del linguaggio fu definito nel 1996 da Grady Booch, Jim Rumbaugh e Ivar Jacobson sotto l'egida dello OMG (Object Management Group), che tuttora gestisce lo standard di UML. Il linguaggio nacque con l'intento di unificare le notazioni di modellazione sviluppate in precedenza, indipendentemente, dai tre padri di UML.
Il linguaggio consente di rappresentare in modello un sistema software orientato agli oggetti secondo numerosi aspetti (funzionali, strutturali, dinamici) e a diversi livelli di dettaglio, con una flessibilità sufficiente a garantire la realizzabilità di modelli accurati sia nella fase di analisi che nelle varie fasi di progetto a diversi livelli di raffinamento, mantenendo la tracciabilità dei concetti impiegati per modellare il sistema in queste varie fasi.

URI Un Universal Resource Identifier è un elemento del protocollo Internet consistente in una stringa di caratteri conforme ad una certa sintassi [21]. La stringa rappresenta un nome, o indirizzo, che può venir utilizzato per identificare univocamente una risorsa. È un concetto fondamentale del World Wide Web. Classico esempio di URI è l'indirizzo di una pagina internet (come http://www.pippo.org) che in questo caso viene chiamato URL (Uniform Resource Locator) ossia una URI che descrive anche il metodo per ottenere una descrizione della risorsa in questione.

Bibliografia

[1] Beckett, D. Rdf/xml syntax specification. URL: http://www.w3.org/TR/rdf-syntax-grammar/, ultima consultazione il 16 Luglio 2005.

[2] Beckett, D. Redland rdf application framework. URL: http://librdf.org, ultima consultazione il 2 Ottobre 2005.

[3] Berners-Lee, T. Primer: Getting into rdf and semantic web using n3. URL: http://www.w3.org/2000/10/swap/Primer, ultima consultazione il 18 Luglio 2005.

[4] Bray, T. What is rdf. URL: http://www.xml.com/pub/a/2001/01/24/rdf.html, ultima consultazione il 13 Marzo 2005.

[5] Dan Brickley, R. G. Rdf vocabulary description language 1.0: Rdf schema. URL: http://www.w3.org/TR/rdf-schema/, ultima consultazione il 16 Luglio 2005.

[6] David C. Fallside, P. W. Xml schema part 0: Primer second edition. URL: http://www.w3.org/TR/xmlschema-0/, ultima consultazione il 20 Luglio 2005.

[7] Deborah McGuinness, F. v. H. Owl web ontology language overview. URL: http://www.w3.org/TR/owl-features/, ultima consultazione il 6 Giugno 2005.

[8] Elliotte Rusty Harold, W. S. M. *XML in a Nutshell*. O'Reilly, 2004.

[9] Eric Prud'hommeaux, A. S. Sparql query language for rdf. URL: `http://www.w3.org/TR/rdf-sparql-query/`, ultima consultazione il 20 Settembre 2005.

[10] Frank Manola, E. M. Rdf primer. URL: `http://www.w3.org/TR/rdf-primer/`, ultima consultazione il 13 Marzo 2005.

[11] Grimes, R. Revolutionary file storage system lets users search and manage files based on content. URL: `http://msdn.microsoft.com/msdnmag/issues/04/01/WinFS/`, ultima consultazione il 18 Luglio 2005.

[12] Inc., A. Apple mac os x: Spotlight. URL: `http://www.apple.com/macosx/features/spotlight/`, ultima consultazione il 18 Luglio 2005.

[13] Inc., S. N. Xml-rpc official homepage. URL: `http://www.xmlrpc.org`, ultima consultazione il 2 Ottobre 2005.

[14] Martelli, A. *Python Cookbook*. O'Reilly, 2002.

[15] Michael K. Smith, Chris Welty, D. L. M. Owl web ontology language guide. URL: `http://www.w3.org/TR/owl-guide/`, ultima consultazione il 6 Giugno 2005.

[16] Mike Dean, G. S. Owl web ontology language reference. URL: `http://www.w3.org/TR/owl-ref/`, ultima consultazione il 17 Settembre 2005.

[17] Miloslav Nic, A. A. Dtd tutorial. URL: `http://www.zvon.org/xxl/DTDTutorial/General\_ita/book.html`, ultima consultazione il 20 Luglio 2005.

[18] Pilgrim, M. *Dive into Python*. Apress, 2004. URL: `http://diveintopython.org`, ultima consultazione il 12 Settembre 2005.

[19] Powers, S. *Practical RDF*. O'Reilly, 2003.

[20] Signore, O. Rdf per la rappresentazione della conoscenza. URL: http://www.w3c.it/papers/RDF.pdf, ultima consultazione il 21 Febbraio 2005.

[21] Tim Berners-Lee, R. Fielding, L. M. Uri: Generic syntax. URL: http://www.ietf.org/rfc/rfc3986.txt, ultima consultazione il 6 Giugno 2005.

[22] Tim Berners-Lee, James Hendler, O. L. The semantic web. URL: http://www.scientificamerican.com/article.cfm?articleID\00048144-10D2-1C70-84A9809EC588EF21&catID2=, ultima consultazione il 10 Giugno 2005.

[23] Tim Bray, Dave Hollander, A. L. Namespaces in xml 1.1. URL: http://www.w3.org/TR/xml-names11/, ultima consultazione il 20 Luglio 2005.

[24] Tim Bray, Jean Paoli, C. S.-M. Xml reference. URL: http://www.xml.it:23456/XML/REC-xml-19980210-it.html, ultima consultazione il 20 Luglio 2005.

[25] Vari, A. Beagle official homepage. URL: http://beaglewiki.org/Main_Page, ultima consultazione il 18 Luglio 2005.

[26] Vari, A. Description of the dublin core elements. URL: http://dublincore.org/documents/dces, ultima consultazione il 5 Settembre 2005.

[27] Vari, A. Dublin core homepage. URL: http://dublincore.org, ultima consultazione il 5 Settembre 2005.

[28] Vari, A. Gnome: The free software desktop project. URL: http://www.gnome.org, ultima consultazione il 2 Febbraio 2005.

[29] Vari, A. Iana mime media types. URL: http://www.iana.org/assignments/media-types ultima consultazione il 10 Luglio 2005.

[30] Vari, A. Imagemagick: Convert, edit and compose images. URL: http://www.imagemagick.org, ultima consultazione il 6 Settembre 2005.

[31] Vari, A. Nautilus file manager. URL: http://www.gnome.org/projects/nautilus, ultima consultazione il 10 Settembre 2005.

[32] Vari, A. Python official homepage. URL: http://www.python.org, ultima consultazione il 2 Ottobre 2005.

[33] Vari, A. Quelcom project page. URL: http://www.etse.urv.es/~dmanye/quelcom/quelcom.html, ultima consultazione il 6 Settembre 2005.

[34] Vari, A. Rdf, accessing collections. URL: http://www.w3.org/2001/sw/DataAccess/issues\#accessingCollections, ultima consultazione il 16 Luglio 2005.

[35] Vari, A. Rdf data access working group. URL: http://www.w3.org/2001/sw/DataAccess/, ultima consultazione il 16 Luglio 2005.

[36] Vari, A. Sed official homepage. URL: http://sed.sourceforge.net, ultima consultazione il 6 Settembre 2005.

[37] Vari, A. Sendemail official homepage. URL: http://www.iana.org/assignments/media-types/, ultima consultazione il 6 Settembre 2005.

[38] Vari, A. W3c official homepage. URL: http://w3.org, ultima consultazione il 2 Ottobre 2005.

[39] Vari, A. wxwidgets official homepage. URL: http://www.wxwidgets.org, ultima consultazione il 2 Ottobre 2005.