Riuso del Software

Riferimenti:

I.Sommerville, Ingegneria del Software, 8° edizione- Cap. 18 (Riuso)

Outline

- Vantaggi del riuso
- Panoramica sulle forme di riuso
- Design Pattern
- Riuso basato su generatori
- Application Frameworks
- Riuso di COTS

Riuso del Software

- Nella maggior parte delle discipline ingegneristiche, I sistemi si progettano a partire da componenti che sono usati anche da altri sistemi.
- L'Ingegneria del Software si è originariamente preoccupata soprattutto dello sviluppo di software ex-novo, ma oggi è ben noto che occorre adottare processi di sviluppo basati su un riuso sistematico del software:
  - per ottenere software migliore e
  - per ottenere software più rapidamente ed economicamente.

Vantaggi del riuso

- Riusare software esistente può consentire di:
  - Ridurre i costi di sviluppo/testing/manutenzione
  - Produrre software che rifletta il livello di affidabilità dei software riusati: in presenza di software riusabili di grande qualità può essere una ottima opzione

- Riusare è possibile a diversi livelli:
  - Riuso di intere applicazioni (es. COTS)
  - Riuso di componenti (v. CBSE)
  - Riuso di *oggetti* e *funzioni*

Benefici del Riuso

- Maggiore affidabilità
  - Il software riusato è, generalmente, più affidabile (in quanto già provato e testato) di quello prodotto ex-novo.
- Minori rischi di progetto
  - Se si può riusare software, piuttosto che svilupparlo ex-novo, si possono fare stime più accurate sui costi del progetto
- Sviluppo più rapido
  - I tempi di sviluppo si accorciano ed è possibile consegnare il software più rapidamente

Problemi legati al riuso

- Maggiori costi di manutenzione
  - Se si riusa software di cui non si possiede il codice, il sistema complessivo potrà risultare meno flessibile e meno manutenibile
- Mancanza di strumenti CASE
  - Spesso i CASE (strumenti a supporto dello sviluppo) non forniscono funzionalità che consentono un efficace riuso di librerie di componenti software
- Diffidenza verso software prodotto da altri
  - Spesso si preferisce risviluppare per avere un maggior controllo sul software prodotto

Problemi legati al riuso

- Difficoltà nel creare e mantenere librerie di componenti riusabili
  - Popolare librerie di componenti riusabili può essere costoso, e le tecniche disponibili per classificare, catalogare e ricercare componenti software non sono ancora mature.
- Difficoltà nel trovare, comprendere ed adattare componenti riusabili
  - Non sempre si è disposti a spendere tempo per cercare, comprendere ed eventualmente adattare un componente riusabile.

Fattori di cui tener conto nel pianificare il riuso

- Tempistica richiesta per lo sviluppo
- Durata prevista per la vita del software
- Background, capacità ed esperienza del team di sviluppo
- Criticità del software e altri requisiti non funzionali
- Dominio di applicazione
- Piattaforma sulla quale eseguire il sistema

1. Riuso a livello Concettuale

- Il Riuso di componenti già implementati obbliga ad ereditare le scelte di progetto e di sviluppo di chi ha realizzato I componenti, limitando le situazione nelle quali il riuso è possibile…

- Ad un maggiore livello di astrazione, è possibile invece riutilizzare “concetti”, ovvero scelte effettuate durante la specifica dei requisiti o la fase di progettazione
  - Es. Riuso di schemi di progettazione usati per Algoritmi fondamentali, tipi di dato astratto;
  - Design patterns;
  - Generative programming.

Design Pattern

- Un Pattern
  - individua un'IDEA, uno schema GENERALE E RIUSABILE
    - schema di problema,
    - schema di soluzione, etc.
  - Rispetto ai componenti riusabili
    - non è un "oggetto" fisico
    - non può essere usato così come è stato definito, ma deve essere contestualizzato all'interno del particolare problema applicativo
  - Due istanze/contestualizzazioni di uno stesso pattern (ad esempio in problemi diversi) tipicamente sono diverse proprio per la contestualizzazione in domini differenti

Scopo dei Patterns

- Scopo dei pattern
  - Catturare l'esperienza e la "saggezza" degli esperti
  - Evitare di reinventare ogni volta le stesse cose
- Cosa fornisce un design pattern al progettista software?
  - Una soluzione codificata e consolidata per un problema ricorrente
  - Un'astrazione di granularità e livello di astrazione più elevati di una classe
  - Un supporto alla comunicazione delle caratteristiche del progetto
  - Un modo per progettare software con caratteristiche predefinite
  - Un supporto alla progettazione di sistemi complessi

Definizione

- Definizione:
  - Ogni pattern descrive un problema specifico che ricorre più volte e descrive il nucleo della soluzione a quel problema, in modo da poter utilizzare tale soluzione un milione di volte, senza mai farlo allo stesso modo.
- Abbastanza astratti
  - in modo da poter essere condivisi da progettisti con punti di vista diversi
- Non complessi nè domain-specific
  - non rivolti alla specifica applicazione ma riusabili in parti di applicazioni diverse

Caratteristiche

- Un Design Pattern Nomina, Astrae, e Identifica
  - gli aspetti chiave di una struttura comune di design che la rendono utile nel contesto del riuso in ambito object-oriented
- Un Design Pattern identifica
  - le classi (e le istanze) partecipanti
  - le associazioni ed i ruoli
  - le modalità di collaborazione tra le classi coinvolte
  - la distribuzione delle responsabilità nella soluzione del particolare problema di design considerato

Come sono fatti i Design Patterns

Un pattern è formato da quattro elementi essenziali:

1. Il **nome** del pattern, è utile per descrivere la sua funzionalità in una o due parole.

2. Il **problema** nel quale il pattern è applicabile.

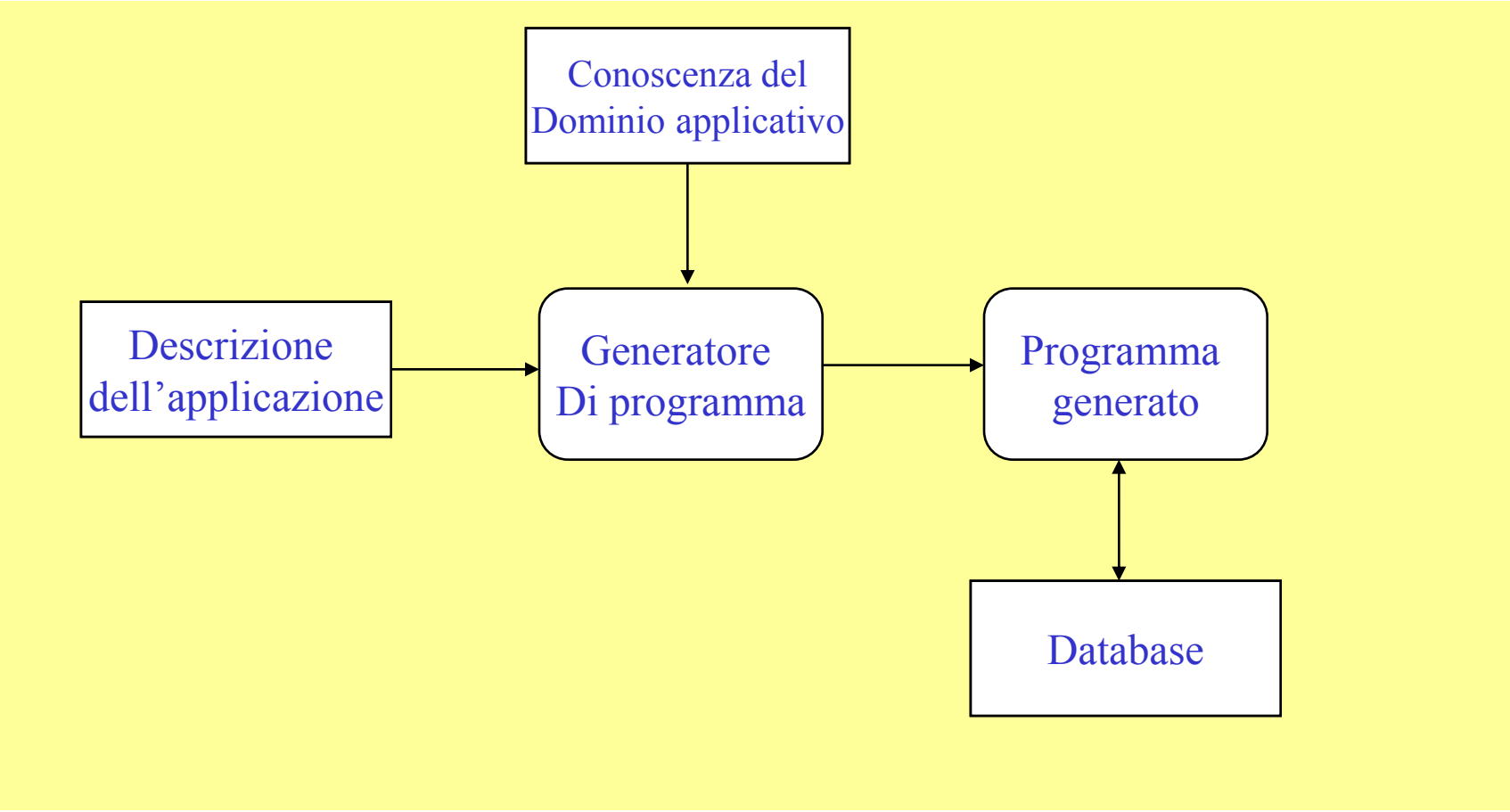
3. La **soluzione** che descrive in modo astratto come il pattern risolve il problema. Descrive gli elementi che compongono il design, le loro responsabilità e le collaborazioni.

4. Le **conseguenze** portate dall'applicazione del pattern. Spesso sono tralasciate ma sono importanti per poter valutare i costi-benefici dell'utilizzo del pattern.

2. Riuso basato su Generatori

- Un generatore è un software che è in grado di generare, a sua volta, software parametrizzato in base a delle specifiche fornite dall'utente
- I generatori possono essere utilizzati nell'ambito di quei problemi per I quali esistono soluzioni ben consolidate che però dipendono notevolmente dai dati in ingresso
- I dati in ingresso al generatore di programmi vanno a descrivere la conoscenza relativa al dominio per il quale debba essere utilizzato il programma da generare

Riuso basato su Generatori



La generazione di un programma attraverso un generatore di codice

Esempi di generatori di codice (1)

- Generatori di applicazioni per gestione dati aziendali
  - I dati di dominio servono semplicemente alla personalizzazione del prodotto;

- Parser e analizzatori lessicali per analisi di codice:
  - L'input è una grammatica del linguaggio da analizzare; l'output è l'analizzatore del linguaggio.
  - Es. Antlr (vedi appendice)
  - Lex&Yacc (per codice C) e JavaCC (per codice Java)

Esempi di generatori di codice (2)

- Generatori di codice basati su modelli
  - Ad esempio componenti software, inclusi in strumenti CASE per la modellazione di software, che generano frammenti di codice a partire da modelli (ad esempio da modelli UML)
  - **DPAToolkit**
    - Genera uno scheletro di codice java/cpp che istanzia un design pattern
      - http://dpatoolkit.sourceforge.net/
  - **Web Ratio**
    - Genera un'applicazione web a partire da un modello codificato in WEBml, che a sua volta estende modelli delle classi e modelli E-R
      - http://www.webratio.com/Home.do?link=oln489d.redirect

Vantaggi e svantaggi dei generatori

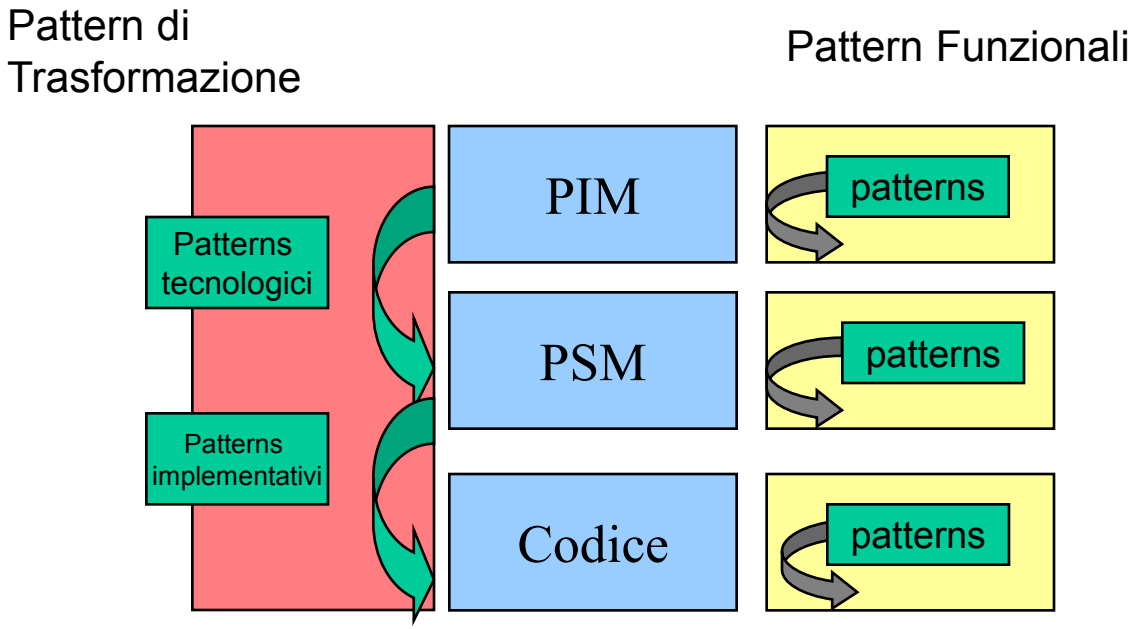
- Il riuso basato su generatori riduce notevolmente il costo di sviluppo e produce codice molto affidabile
- Tramite generatori di codice si possono ottenere programmi più versatili e performanti di quanto si possa ottenere limitandosi a leggere direttamente dal database, a tempo di esecuzione, I dati relativi alla personalizzazione
  - Non tutti i generatori di codice, però, sono in grado di generare codice che sia anche efficiente
- Scrivere una descrizione di dominio per un utente programmatore è più semplice che sviluppare programmi da zero (anche se lo skill richiesto non è minimo)

- La loro applicabilità si limita a poche tipologie di problemi
- Spesso il linguaggio col quale descrivere il problema al generatore ha una semantica molto limitata

Model Driven Architectures (MDA)

- Famiglia di standard di modellazione (basata su UML e standardizzata da OMG), pensati allo scopo di generare codice eseguibile a partire da modelli
  - MDA si basa sulla separazione fra livelli di astrazione (dominio, tecnologia, codice)
  - MDA si basa sulla automazione della trasformazione fra modelli di diverso livello
    - Ad ogni modello previsto da MDA deve corrispondere una famiglia di strumenti che attuino le regole di traduzione previste per passare da modelli **PIM** (Platform Independent Model) verso modelli **PSM** (Platform Specific Model), e dal PSM verso il **codice**.

MDA- separazione fra livelli e trasformazioni



La separazione fra modelli PIM, PSM e codice e la traduzione fra modelli attraverso vari pattern

Speranze e difficoltà

- La speranza per il futuro è che i generatori di codice possano un giorno tradurre dal modello di progetto al codice allo stesso modo di come, oggi, i compilatori traducono da linguaggi di alto livello ad assembler

- Le difficoltà sono legate soprattutto alla formalizzazione di linguaggi di progettazione dotati di semantica univoca e sufficiente alla trasformazione univoca verso codice di alto livello (o eseguibile)

3. Application Framework

- I framework rappresentano modelli astratti di progetto di sotto-sistemi. Le applicazioni si costruiscono integrando e completando una serie di framework.

- Es.: OO Framework
  - Composti di una collezione di classi astratte e concrete e di interfacce tra loro;
  - Un framework OO è una struttura generica; per realizzare il software bisogna riempire le parti del progetto instanziando le classi astratte necessarie, ed implementando il codice mancante

- Si differenziano dai design patterns per il fatto di essere astrazioni di livello più alto, a livello architetturale anzichè di design.

Framework

- Spesso i framework sono istanziazioni di una serie di design pattern

- L'utilizzo di un framework da parte di programmatori e progettisti comporta il raggiungimento di un notevole skill riguardante la conoscenza della struttura del framework e delle opportunità da esso messe a disposizione.
  - Spesso è necessario molto tempo, prima di poter maturare tali conoscenze

Esempi di framework

- Struts
- Spring
- …
- Android Development Toolkit (ADT)
- …
- JUnit

4. Riutilizzo di intere applicazioni

- Consiste nel riutilizzare, eventualmente previa riconfigurazione o personalizzazione, intere applicazioni.
- Le applicazioni possono essere riutilizzate direttamente o essere integrate fra loro come componenti indipendenti all'interno di più ampi sistemi.
- Esistono due approcci principali:
  - *Integrazione di COTS;*
  - *Sviluppo di Linee di Prodotti*

Riuso di COTS

- COTS - Commercial Off-The-Shelf- Software commerciale che può essere usato dai suoi acquirenti senza modifiche (es. Soluzioni desktop, prodotti server…)
  - Esempio storico di COTS sono i DBMS
- Si tratta di applicazioni complete che spesso offrono un API (Application Programming Interface) per permettere ad altri componenti software di accedere alle proprie funzionalità
- Nella pratica, i COTS sono composti di un insieme di classi astratte di interfaccia visibili all'esterno e di un insieme di classi astratte e concrete non visibili

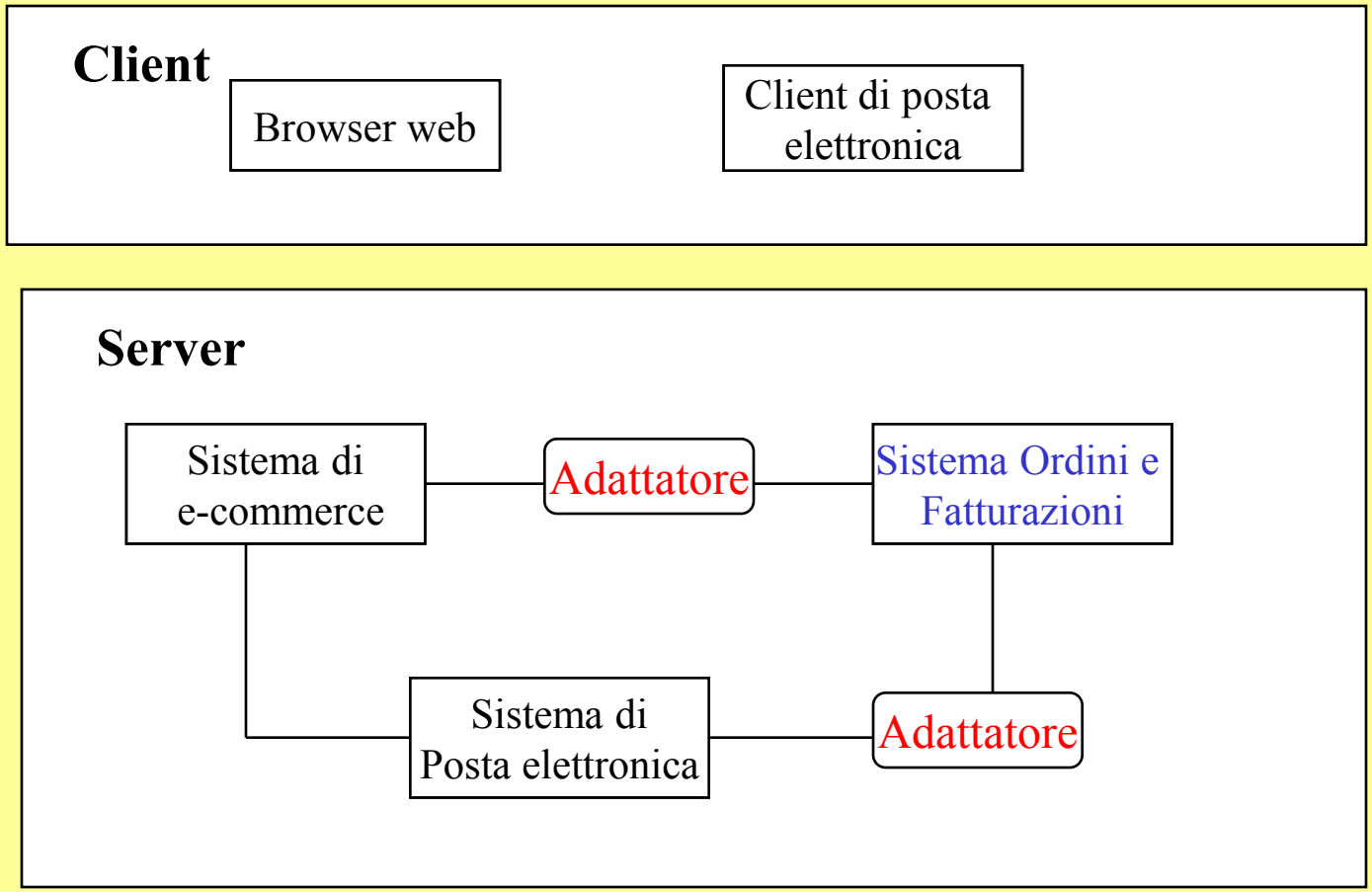
Integrazione di COTS

- Costruire grandi sistemi integrando COTS è una strategia abbastanza efficiente per sistemi le cui funzionalità base siano abbastanza comuni
  - ad esempio per realizzare sistemi di E-Commerce, potrebbero essere riutilizzabili COTS che implementano soluzioni ai problemi di autenticazione, gestione del database, invio delle e-mail, browsing delle pagine web, etc.
- Tramite COTS si velocizza il processo di sviluppo riducendo i costi di sviluppo e test

Esempio: realizzare un sistema di acquisto basato su COTS

- Una organizzazione vuole dotarsi di un sistema per consentire ai suoi dipendenti di effettuare ordini dai propri PC
- L'organizzazione possiede già un sistema COTS per l'ordinazione ed uno per la fatturazione e consegna (usato solo dall'ufficio ordini). Si sceglie di costruire il nuovo sistema integrando tali COTS.
- Sul lato client, saranno usati programmi standard di e-mail e web browsing.
- Sul server, si integrerà una piattaforma per l'e-commerce con I due sistemi esistenti.
  - Saranno necessari degli adattatori per consentire lo scambio di dati.
  - Verrà inoltre usato un sistema di e-mail per generare e-mail di notifica e ci sarà bisogno di un altro adattore per ricevere dati dal sistema di fatturazione.

Esempio: Sistema Acquisti basato su COTS



Un esempio di sistema basato su COTS

Problemi dei COTS

- **Mancanza di controllo sulle funzionalità e sulle prestazioni**
  - Il sistema è utilizzato a scatola chiusa: non siamo consapevoli di come avvengano realmente le operazioni nel suo interno e, di conseguenza, non siamo in grado di modulare le richieste in modo da ottimizzare le prestazioni.
- **Problemi di interoperabilità tra sistemi COTS**
  - Può essere necessario dover scrivere del codice extra per integrare sistemi COTS di produttori indipendenti.
- **Nessun controllo sull'evoluzione del sistema**
  - Può avvenire che le nuove versioni del sistema rendano impossibile l'interazione col nostro software costringendoci a rimanere legati alle vecchie versioni, per le quali non c'è più manutenzione.
- **Supporto dei produttori COTS**
  - Tipicamente il supporto dei produttori potrebbe terminare con l'acquisto del prodotto o limitarsi alle sole evoluzioni decise dal produttore.

5. Linee di prodotti software

- Per linee di prodotti software si intendono famiglie di applicazioni con *funzionalità generiche* che si prestano ad essere configurate o adattate in modo da poter essere utilizzate in contesti specifici.

- Esempi di adattamento:
  - Specifiche configurazioni dei componenti e del sistema;
  - Aggiunta di nuovi componenti;
  - Selezione di componenti nell'ambito di una libreria;
  - Modifiche ai componenti per adattarsi alle esigenze del contesto.

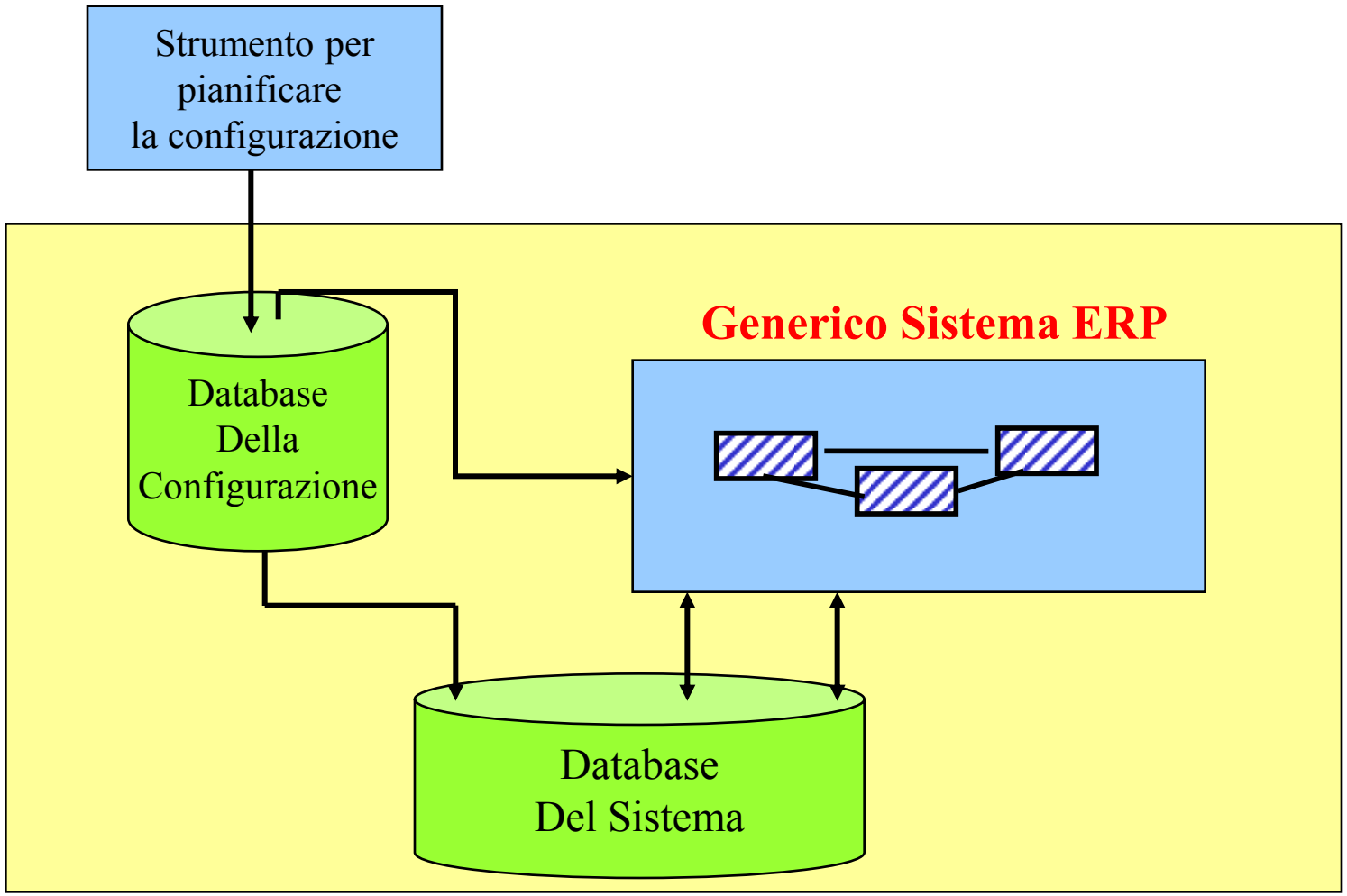
Configurazione

- Può avvenire in due momenti diversi:
- **Configurazione alla consegna**
  - La configurazione avviene per un prodotto finito, senza modificarne internamente la struttura e il progetto, ma solo limitandone/personalizzando le funzionalità (customizzazione).
    - Es. Pacchetti software verticali (es. ERP)
- **Configurazione a tempo di progettazione**
  - Tramite l'adozione di patterns e framework generici, le richieste di personalizzazione vengono recepite in fase di progetto e influiscono direttamente sulla realizzazione del prodotto

Esempio: sistemi ERP

- **Enterprise Resource Planning** (ERP): sistema (generico) che supporta comuni processi aziendali (quali gestione ordini, fatture, inventari, paghe) (es. SAP e BEA)
- Il processo di configurazione degli ERP si basa sull'adattamento di un core generico attraverso l'inclusione e la configurazione di moduli, e incorporando conoscenza su processi e regole aziendali del cliente specifico in un database di sistema.
- Molto usati in grandi aziende, costituiscono la forma di riuso più comune.

Organizzazione di un Sistema ERP



Appendice: ANTLR

Bibliografia

- Terence Parr, The Definitive ANTLR Reference: Building Domain-Specific Languages, The Pragmatic Programmers, http://www.pragprog.com/titles/tpantlr
- http://www.antlr.org/

ANTLR

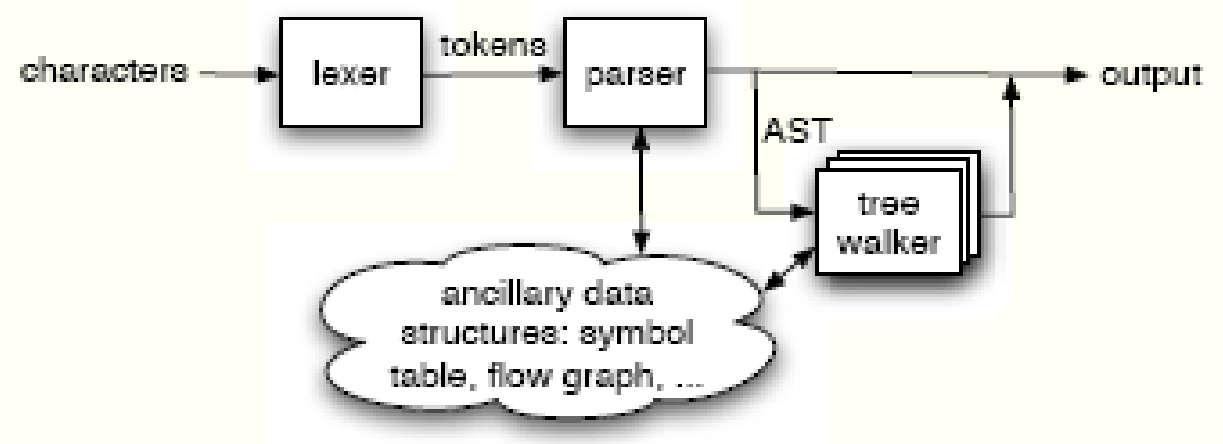
- ANTLR (Another Tool for Language Recognition) è un sofisticato generatore di parser che può essere utilizzato per implementare riconoscitori, compilatori, traduttori, strumenti di misura di metriche di prodotto per qualsiasi linguaggio, in particolare per I cosiddetti DSL (domain-specific languages)
  - Esempi classici di DSL sono file dati, I file di configurazione, I protocolli di rete e molti altri linguaggi specifici di un dominio.

- L'utilizzo di Antlr non è consigliabile per quei linguaggi che sono dei dialetti di XML, in quanto, per essi, è più conveniente riutilizzare le librerie per il parsing XML

Traduttori

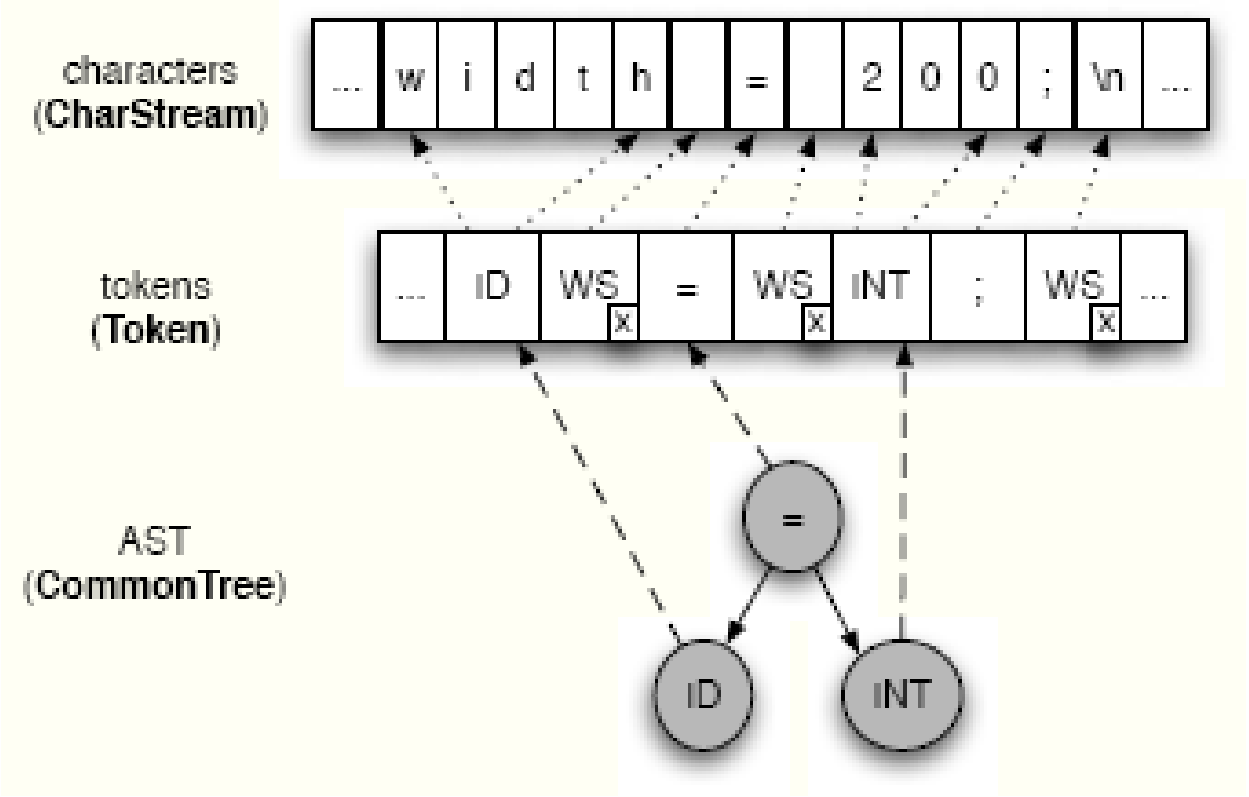
- Un traduttore lavora su di un documento in input, scritto in un certo linguaggio, e traduce ogni frase in un'altra nel linguaggio di output.
- Per realizzare questa trasformazione, esso esegue del codice sorgente realizzato ad hoc per il inguaggio da analizzare
- Un traduttore dovrà quindi esibire comportamenti diversi per ognino delle possibili tipologie di frasi (sentences) trovate nel documento in input.

Il processo di traduzione



- Il Lexer riconosce tokens (ad esempio keywords) dall'analisi del testo (analisi lessicale);
- Il parser riconosce strutture formate da token (analisi sintattica)
- L'uscita più efficiente da gestire è quella in forma di AST (Abstract Syntax Tree)

Esempio


characters
(CharStream)
... w i d t h = 2 0 0 ; \n ...
tokens
(Token)
... ID WS = WS INT ; WS ...
AST
(CommonTree)
=
ID
INT
Lexer
Parser

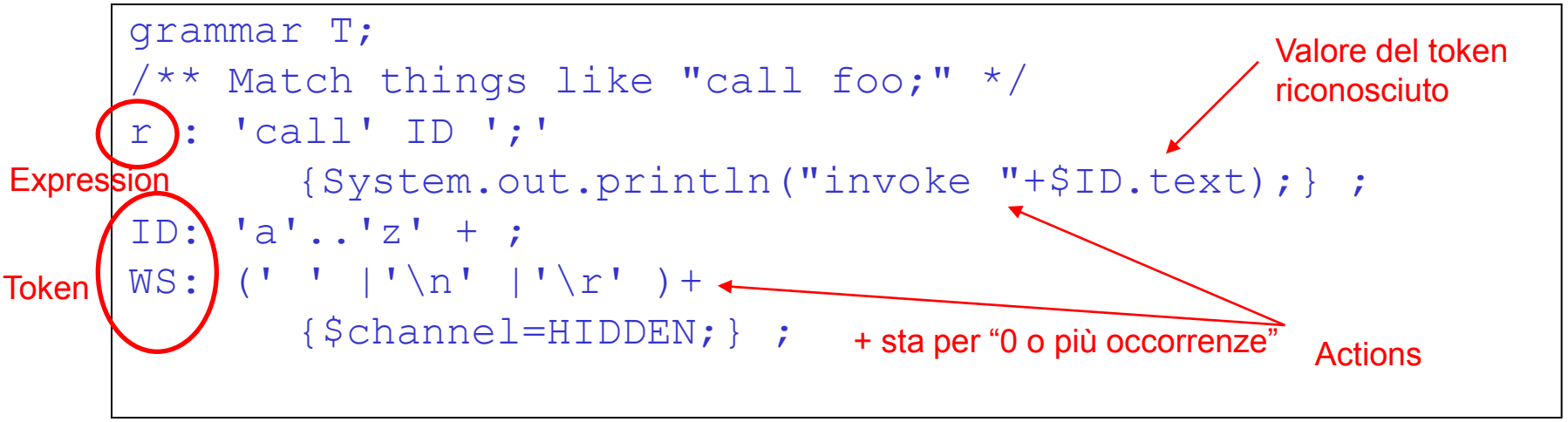
In dettaglio

- La prima fase della traduzione è l'analisi lessicale, che opera direttamente sullo stream in ingresso, riconoscendo *tokens* a partire dall'analisi dei caratteri dello stream di input
- La seconda fase è il parsing propriamente detto, che elabora i tokens restituiti dal lexer, riconosce la loro organizzazione sintattica ed esegue delle elaborazioni conseguenti al loro riconoscimento
  - ANTLR è in grado di generare automaticamente il codice sorgente dell'analizzatore lessicale e del parser sulla base della descrizione della grammatica e delle elaborazioni conseguenti ai riconoscimenti dei token e delle strutture (che devono essere forniti dall'utente)

Riassumendo …

- L’analizzatore lessicale organizza in token lo stream di input
- Il parser legge questa sequenza di token e cerca di riconoscere le frasi e la loro struttura
- Il più semplice traduttore si limita a proporre in output le frase riconosciture, senza ulteriori fasi
- Traduttori più complessi costruiscono strutture organizzative come gli AST, per consentire un’agevole interpretazione ad altri software da posizionare più a valle

Esempio (banale) di grammatica




```
$ java org.antlr.Tool T.g
ANTLR Parser Generator Version 3.0 1989-2007
$ ls
T.g TLexer.java T__.g T.tokens TParser.java
```

Main di test

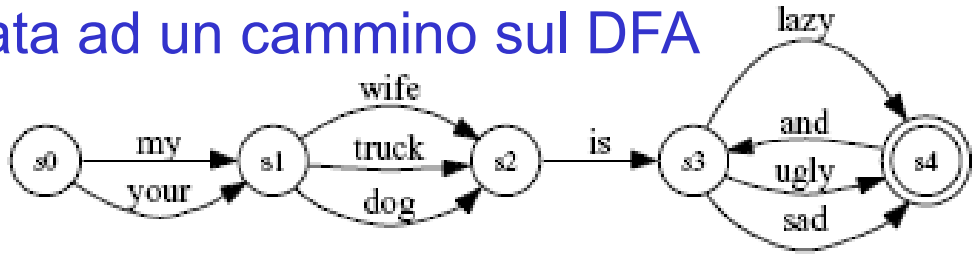
```
import org.antlr.runtime.*;
public class Test {
   public static void main(String[] args) throws Exception {
   // create a CharStream that reads from standard input
    ANTLRInputStream input = new ANTLRInputStream(System.in);
    // create a lexer that feeds off of input CharStream
    TLexer lexer = new TLexer(input);
    // create a buffer of tokens pulled from the lexer
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    // create a parser that feeds off the tokens buffer
    TParser parser = new TParser(tokens);
    // begin parsing at rule r
    parser.r();
    }
}
```

```
$ javac TLexer.java TParser.java Test.java
$ java Test
call foo; EOF
invoke foo
```

State Machines

- Gli automi deterministici a stati finiti (DFA) possono essere utilizzati come generatori di espressioni regolari

- Nell'esempio, ogni percorso dallo stato iniziale a quello finale corrisponde ad una delle possibili espressioni regolari
  - Quindi, un'espressione è regolare, se può essere associata ad un cammino sul DFA

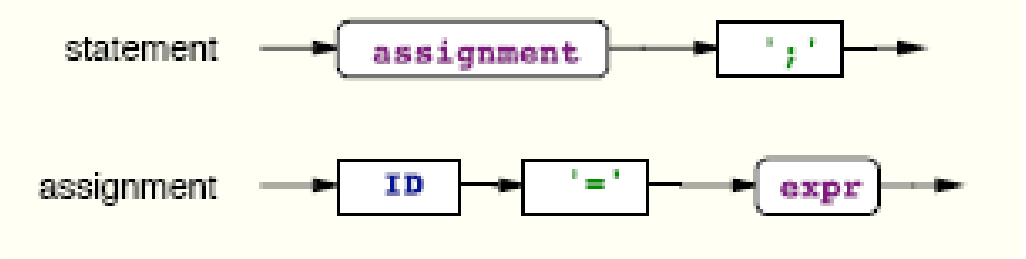


PushDown Automata

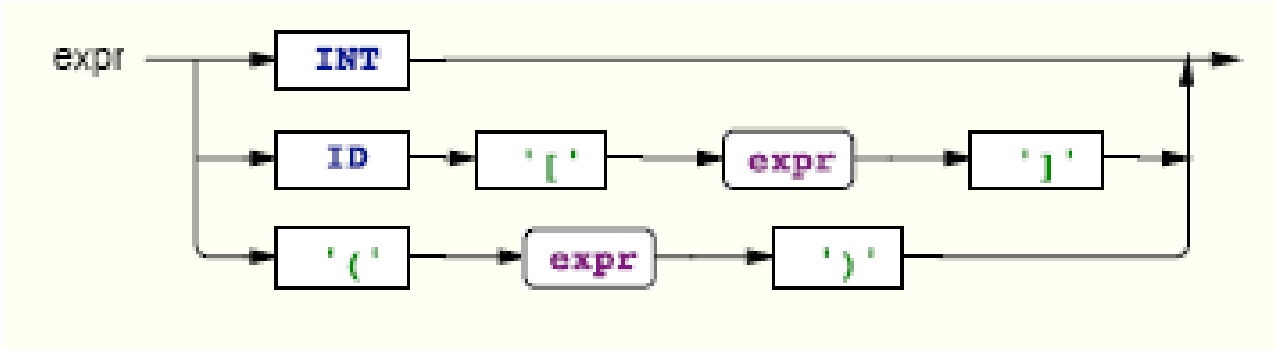
- Le macchine a stati hanno un comportamento che non dipende dalla “memoria” della macchina
  - In pratica bisogna considerare stati diversi per ogni possibile valore della memoria
    - Il numero di stati andrebbe ad esplodere
- Per questo motivo si preferiscono i Pushdown Automata
  - Si tratta di Automi a Stati Finiti dotati di memoria a stack
  - Un Pushdown Automata, essendo dotato di memoria, può essere visto come un insieme di submachine
    - Le transizioni tra uno stato e l’altro dipendono, oltre che dagli eventi che si verificano, anche dal valore di variabili memorizzate nello stack
  - Graficamente si indicano tramite Syntax Diagrams

Syntax Diagram

- Syntax Diagram



- Syntax Diagram ricorsivo



Left Lookahead (LL)

- Antlr supporta un riconoscimento delle frase di tipo Left Lookahead (LL)
  - In pratica, data un'espressione come

```
decl : 'int' ID '=' INT ';' // E.g., "int x = 3;"
                | 'int' ID ';' // E.g., "int x;"
                ;
```

    È possibile per il parser distinguere tra la prima e la seconda regola solo andando a vedere cosa accade 2 parole più avanti rispetto alla parola 'int' (la presenza dell' '=' anzichè del ';')
  - In questo caso è necessario un parser LL(2)
- Antlr è in grado di generare parser che siano LL(*)

Un esempio completo

- Si vogliono riconoscere e calcolare espressioni di assegnazione tali che:
  - sul lato destro ci sono espressioni aritmetiche con gli operatori di somma e prodotto e l'uso delle parentesi per esprimere precedenza
  - sul lato sinistro (opzionale) ci sono variabili cui assegnare il valore calcolato sul lato destro

- Esempi:

  a=3 (memorizza 3 in a)

  b=4 (memorizza 4 in b)

  2+a*b (restituisce 14)

Obiettivi

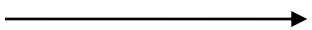
- Gli obiettivi da perseguire sono due:

  1. Costruire una grammatica che descriva la struttura sintattica delle espressioni e delle assegnazioni

  2. Scrivere del codice, contestualmente alla grammatica, che esegue le corrette azioni in conseguenza del riconoscimento delle espressioni della grammatica stessa (ad esempio effettuare una somma quando si riconosce un'espressione con +, e così via)

Grammatica 1/2

```
grammar Expr;

prog:   stat+ ;

stat:   expr NEWLINE
    |   ID '=' expr NEWLINE
    |   NEWLINE
    ;

expr:   multExpr (('+'|'-') multExpr)*
    ;

multExpr
    :   atom ('*' atom)*
    ;
```



"un prog è un insieme di (0 o più) stat (statements)"

"una stat è una expr oppure un ID seguito da '=' e da una expr oppure una riga vuota"

"una expr è data da uno o più coppie di multExpr con in mezzo '+' o '-'"

"una multExpr è data da uno o più coppie di atom con in mezzo '*'"

Grammatica 2/2

```
atom:    INT
     |   ID
     |   '(' expr ')'
     ;
```

"una atom è data da un INT oppure un ID oppure una expr tra parentesi"

Se si incontra un WS (Whitespace) lo si può saltare senza fare nulla (funzione skip())

```
// TOKEN
ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t'|'\n'|'\r')+ {skip();} ;
// END:tokens
```

"ID è dato da zero o più caratteri alfabetici;
INT è dato da zero o più cifre;
NEWLINE è il codice ASCII \n, eventualmente preceduto da \r
WS è un carattere di spazio, tabulazione o da capo, isolato"

Testing della grammatica

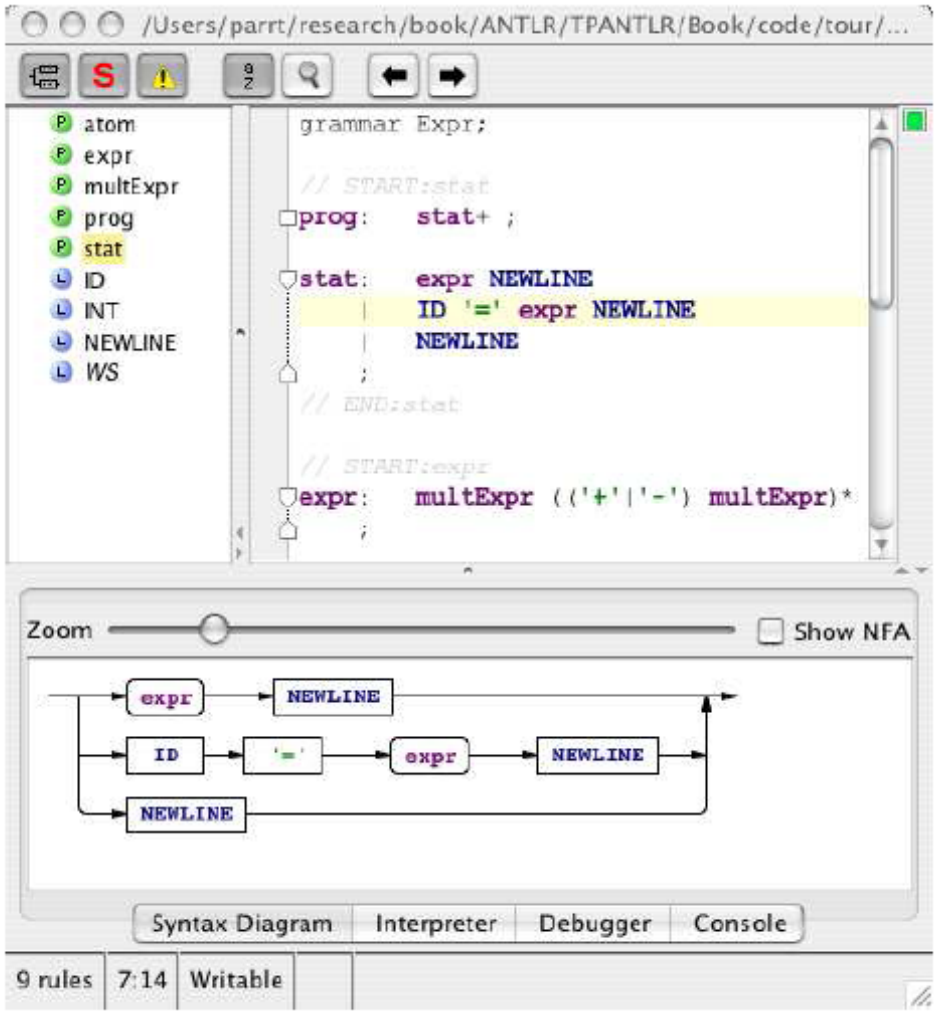
```
import org.antlr.runtime.*;

public class Test {
    public static void main(String[] args) throws Exception {
        // Create an input character stream from standard in
        ANTLRInputStream input = new ANTLRInputStream(System.in);
        // Create an ExprLexer that feeds from that stream
        ExprLexer lexer = new ExprLexer(input);
        // Create a stream of tokens fed by the lexer
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        // Create a parser that feeds off the token stream
        ExprParser parser = new ExprParser(tokens);
        // Begin parsing at rule prog
        parser.prog();
    }
}
```

Questa classe di test consente di elaborare lo stream di input proveniente da System.in

L'unico output possibile, per ora è costituito dalle eccezioni che possono rilevare parser o lexer

Antlr Works

- Dal sito web di ANTLR, è possibile scaricare anche Antlr Works, un tool visuale che supporta la scrittura di grammatiche per antl
- http://www.antlr.org/works
- AntlrWorks è un ambiente GUI che integra al suo interno anche Antlr



AntlrWorks

- AntlrWorks ha numerose funzioni che semplificano la realizzazione e il testing di grammatiche:
  - Visualizzazione/modifica Syntax Diagram
  - Generazione ed editing di Parser e Lexer
  - Debugging delle grammatiche
    - Con visualizzazione grafica dell'AST trovato
- Tutte le prove successive verranno effettuate tramite AntlrWorks

Prove della grammatica

- L'input:

  a=3

  b=4

  3+(a*b)

  Genera un output vuoto (abbiamo scritto solo il riconoscitore, ma nessuna azione è stata associata)

- L'input:

  3+@

  Genera il seguente output:

  line 1:2 no viable alternative at character '@'

Grammatica con azioni 1/2

```
grammar Expr;

@header { import java.util.HashMap; }

@members { HashMap memory = new HashMap();}

prog:   stat+ ;

stat:  expr NEWLINE {System.out.println($expr.value);}
    |   ID '=' expr NEWLINE
         {memory.put($ID.text, new Integer($expr.value));}
    |   NEWLINE
    ;

expr returns [int value]
    :   e=multExpr {$value = $e.value;}
        (   '+' e=multExpr {$value += $e.value;}
        |   '-' e=multExpr {$value -= $e.value;}
        )*
    ;
```

Inclusioni necessarie

Istanzia una variabile memory

Gli elementi riconosciuti sono indicati con $elemento

$ID.text è il nome dell'elemento ID

Value è un attributo definito per questo ed altri elementi

Grammatica con azioni 2/2

```
multExpr returns [int value]
    :   e=atom {$value = $e.value;} ('*' e=atom {$value *= $e.value;})*
    ;

atom returns [int value]:
   INT {$value = Integer.parseInt($INT.text);}
    |   ID
        {
        Integer v = (Integer)memory.get($ID.text);
        if ( v!=null ) $value = v.intValue();
        else System.err.println("undefined variable "+$ID.text);
        }
   |   '(' expr ')' {$value = $expr.value;}
    ;
ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t'|'\n'|'\r')+ {skip();} ;
```

Converte l'INT trovato in un intero

Sostituisce la variabile col suo valore

Esercizi proposti

- Dato un qualsiasi linguaggio di programmazione (ad esempio C):

1. (banale) scrivere un parser che sia in grado di contare il numero di LOC di un programma
2. Scrivere un parser che sia in grado di valutare la complessità ciclomatica di un metodo