Domain Specific Language per l'Ingegneria dei Sistemi Distribuiti

Prof. Federico Bergenti

federico.bergenti@unipr.it

Domain-Specific Language

- I Domain Specific Language (DSL) sono una tecnologia nuova e in rapida evoluzione
  - Non esiste ancora una definizione accettata per il termine
- Una possibile definizione informale può essere *"un DSL è un linguaggio creato appositamente per supportare la l'analisi e la risoluzione di problemi in un fissato dominio applicativo"*
- Al contrario, un General Purpose Language (GPL) è utile in molti domini applicativi
  - Ma tipicamente non consente di sfruttare le peculiarità del dominio applicativo considerato

Esempi di DSL di Uso Comune

- Alcuni DSL sono di uso molto comune
  - HTML è un DSL per la descrizione di ipertesti
  - PDF è un DSL per la descrizione di pagine
  - SQL è un DSL per la manipolazione di basi di dati relazionali
  - Tutti gli IDL sono linguaggi per la descrizione di interfacce di interoperabilità tra moduli software
  - …
- Ognuno di questi DSL
  - Esiste in più versioni con le relative varianti (*dialetti*)
  - Ha una semantica accettata, anche se spesso informale

Alcune Note Importanti sui DSL

- Un DSL non è necessariamente un linguaggio di programmazione
  - È più spesso un linguaggio di descrizione (*modeling*) per uno specifico dominio applicativo
- Il confine tra DSL e GPL è labile perché dipende dall'ampiezza del dominio applicativo
  - PostScript è un DSL ma è anche un linguaggio di programmazione Turing completo
  - Alcuni linguaggi, come Perl, sono nati come DSL ma poi sono stati usati come GPL
- L'adozione di un DSL per l'analisi o la risoluzione di un problema ha un costo e non sempre risulta conveniente

DSL e Ingegneria (del Software) (I)

- I DSL sono spesso usati per produrre artefatti nell'ambito dei processi di ingegneria (del software)
- Questi artefatti sono
  - Sempre testuali
  - Sempre a un elevato livello di astrazione
  - Spesso usati per specificare requisiti
  - Spesso usati per produrre altri artefatti mediante traduzione automatica

```
-- A behavioral design
-- of an AND port

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ANDPort is
  port(
    x: in STD_LOGIC;
    y: in STD_LOGIC;
    o: out STD_LOGIC);
end ANDPort;
```

DSL e Ingegneria (del Software) (II)

- I DSL sono spesso usati per produrre artefatti nell'ambito dei processi di ingegneria (del software)
- Questi artefatti sono
  - Sempre testuali
  - Sempre a un elevato livello di astrazione
  - Spesso usati per specificare requisiti
  - Spesso usati per produrre altri artefatti mediante traduzione automatica

```
architecture V1 of
  ANDPort is
begin
  process(x, y)
  begin
      if x='1' and y='1'
      then
        o <= '1';
      else
        o <= '0';
      end if;
  end process;
end V1;
```

Model-Driven Engineering

- La Model-Driven Engineering (MDE) raccoglie le metodologie e gli strumenti che assumono che i processi vengano guidati da modelli di dominio
  - Un modello di dominio è un modello *concettuale* di alcuni aspetti di uno specifico dominio applicativo
- La MDE si concentra sulle rappresentazioni astratte e concettuali di tutti gli aspetti di un dominio applicativo
  - Mediante vari modelli, che utilizzano notazioni anche diverse
  - Mettendo in secondo piano le soluzioni algoritmiche ai problemi
- La MDE tende a
  - Concentrarsi *sul problema più che sulla soluzione*
  - Privilegiare approcci basati sulle *trasformazioni* di artefatti

DSL e MDE (I)

- Tipicamente i modelli di dominio sono descritti mediante *notazioni grafiche*
  - Modelli UML statici e dinamici
  - Modelli grafici con notazioni legate al dominio applicativo
  - Modelli testuali di vario tipo, anche in linguaggio naturale
- I DSL offrono la possibilità di introdurre una notazione testuale specifica per modelli di dominio
  - Potenzialmente dotati di una grammatica efficace
  - Potenzialmente dotati di strumenti a corredo che ne rendano efficace l'uso, ad esempio supportando trasformazioni automatiche

DSL e MDE (II)

- Oltre alle caratteristiche già discusse, i DSL utili per la MDE devono sempre avere
  - Una grammatica ben definita
  - Una semantica, spesso informale, ma che permetta di generare automaticamente artefatti utili
  - Strumenti a supporto solidi ed efficaci
- In più, devono essere utili nella comunicazione con gli esperti del dominio

DSL e XML (I)

- XML, insieme a tutte le tecnologie e gli strumenti che lo supportano, rappresenta un modo efficace per costruire DSL
  - XML permette di introdurre nuovi tag legati al dominio applicativo
  - Con i CSS si può fornire una veste grafica efficace
  - Con XSLT è possibile generare artefatti utili
  - …

```
<?xml version="1.0" ?>
<catalog>
  <cd>
    <title>Empire
      Burlesque</title>
    <artist>Bob Dylan</artist>
  </cd>
  <cd>
    <title>Hide your
      heart</title>
    <artist>Bonnie
      Tyler</artist>
  </cd>
  …
</catalog>
```

DSL e XML (II)

- XML, insieme a tutte le tecnologie e gli strumenti che lo supportano, rappresenta un modo efficace per costruire DSL
  - XML permette di introdurre nuovi tag legati al dominio applicativo
  - Con i CSS si può fornire una veste grafica efficace
  - Con XSLT è possibile generare artefatti utili
  - …

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
…	…

DSL e XML (III)

- XML vincola molto la grammatica dei DSL che permette di realizzare
  - Anche se dotati di tag specifici, i testi XML sono tipicamente lontani dal linguaggio degli esperti del dominio
  - La grammatica è sempre *non contestuale*
- A causa di varie limitazioni di tecnologie e strumenti
  - Le estensioni esprimibili sono limitate
  - I vincoli sintattici esprimibili sono limitati
  - Le trasformazioni possibili sono limitate

DSL e Xtext (I)

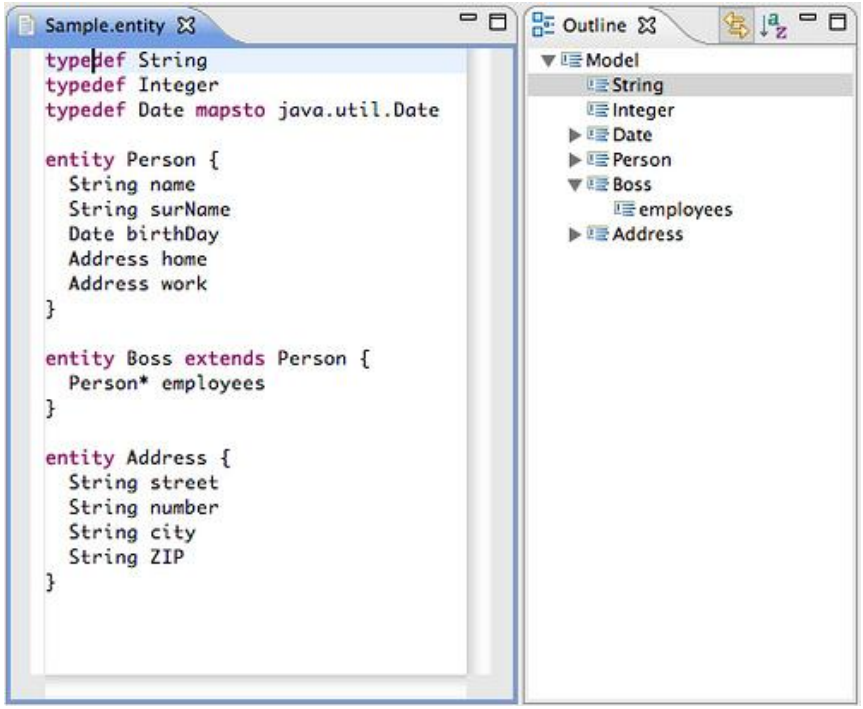
- La suite Xtext è un insieme di strumenti per il supporto alla creazione di DSL
  - Permette di descrivere la sintassi dei DSL mediante *ANother Tool for Language Recognition (ANTLR)*
  - Genera un framework per la realizzazione di traduttori e interpreti
  - Genera vari componenti per Eclipse
  - È integrato con *Eclipse Modeling Framework (EMF)*

eclipse.org/Xtext

www.antlr.org

DSL e Xtext (II)

```
grammar org.xtext.example.Entity
  with org.eclipse.xtext.common.Terminals

generate entity "http://..."

Model : (types += Type)*;

Type : TypeDef | Entity;

Entity : "entity" name=ID
   ("extends" superEntity=[Entity])?
   "{" (attributes+=Attribute)* "}";

Attribute : type=[Type] (many?="*")?
   name=ID;

TypeDef : "typedef" name=ID ("mapsto"
     mappedType=JAVAID)?;

JAVAID : name=ID("." ID)*;
```



La generazione dei componenti per Eclipse è automatica

Sistemi Distribuiti basati su Agenti

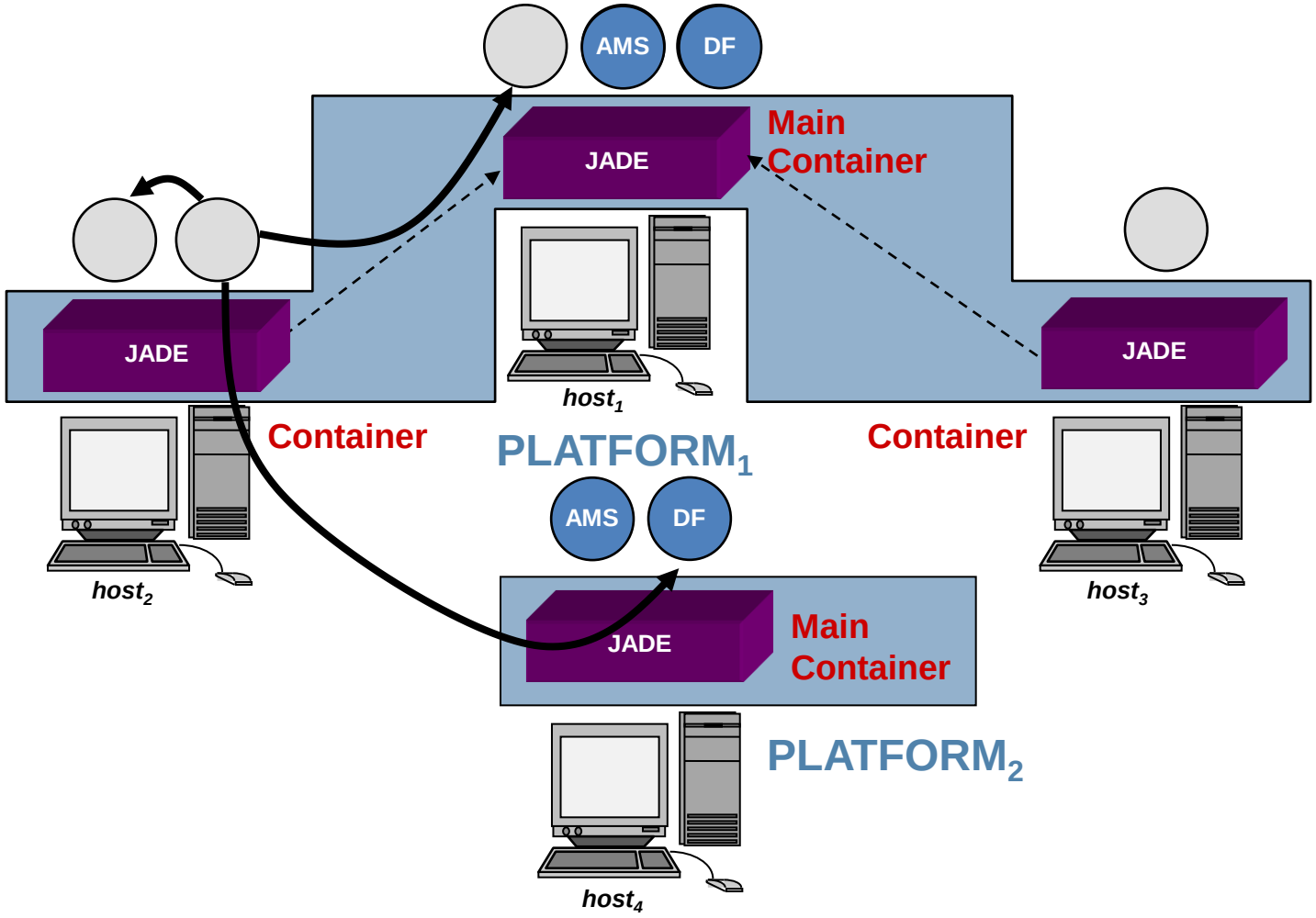
- La tecnologia degli agenti software rappresenta una valida opzione per la realizzazione di sistemi distribuiti
- Lo strumento attualmente più utilizzato in ambito accademico e industriale* è il Java Agent DEvelopment framework (JADE)
  - Una *library* che permette di accedere ai servizi offerti (message passing, naming service, …)
  - Una *supporto a runtime* che offre un ambiente di esecuzione e rende disponibili i servizi al sistema multi-agente

jade.tilab.com

www.fipa.org

* K. Kravari and N. Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015

Architettura di una Piattaforma



Le Astrazioni di JADE

- JADE offre cinque astrazioni *agent-oriented* principali
  - Agente: modulo software atomico che viene attivato in un *container*
  - Behaviour: attività che può essere svolta da un agente (*task*)
  - Messaggio e *ontologia*: utilizzati dagli agenti per comunicare e conformi alla specifica IEEE FIPA
  - Interaction protocol: struttura che descrive sequenze valide di messaggi
  - Servizio a supporto degli agenti: directory service, …
- Le altre astrazioni offerte da JADE non fanno parte del *meta-modello* dei sistemi multi-agente
  - Container, main container, piattaforma, codec dei messaggi, scheduler dei behaviour, …

JADE per il Programmatore Java (I)

```
public class SimpleAgent extends Agent {
  protected void setup() {
    addBehaviour(
      new CyclicBehaviour(this) {
        public void action() {
          System.out.println("Alive");
        }
    });

    addBehaviour(
      new FourStepBehaviour());
  }

  protected void takeDown() {
    System.out.println("About to die");
  }
}
```

```
public class FourStepBehaviour
  extends Behaviour {
  private int step = 1;

  public void action() {
    switch(step) {
    case 1: … break;
    case 2: … break;
    case 3: … break;
    case 4: … break;
    }

    step++;
  }

  public boolean done() {
    return step == 5;
  }
}
```

JADE per il Programmatore Java (II)

```
public class PingAgent extends Agent {
  private MessageTemplate template = and(MatchPerformative(QUERY_IF), MatchOntology("presence"));

  protected void setup() {
    addBehaviour(new CyclicBehaviour(this) {
      public void action() {
        ACLMessage msg = myAgent.receive(template);

        if(msg != null) {
          ACLMessage reply = msg.createReply();

          reply.setContent(msg.getContent());

          if("alive".equals(msg.getContent()))
            reply.setPerformative(ACLMessage.INFORM);
          else
            reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);

          myAgent.send(reply);
        } else block();
      }
    });
  }
}
```

JADE per il Programmatore Java (III)

```
public class EmploymentOntology extends Ontology {
  public static final String NAME = "employment-ontology";
  public static final String ADDRESS = "ADDRESS";
  public static final String ADDRESS_NAME = "street";
  public static final String ADDRESS_NUMBER = "number";
  public static final String ADDRESS_CITY = "city";
  …

  private EmploymentOntology() {
    super(NAME, BasicOntology.getInstance());
    try {
      add(new ConceptSchema(ADDRESS), Address.class);
      ConceptSchema cs = (ConceptSchema)getSchema(ADDRESS);
      cs.add(ADDRESS_NAME, (PrimitiveSchema)getSchema(STRING));
      cs.add(ADDRESS_NUMBER, (PrimitiveSchema)getSchema(INTEGER), OPTIONAL);
      cs.add(ADDRESS_CITY, (PrimitiveSchema)getSchema(STRING), OPTIONAL);
      …
    } catch(OntologyException oe) { … }
  }
}
```

JADE e la AOP

- Tradizionalmente, i sistemi basati su agenti sono realizzati mediante strumenti di Agent-Oriented Programming (AOP)
  - Il più classico linguaggio di AOP è *Agent0*
  - Agent0 è un linguaggio di programmazione *imperativa* ed *event driven*

- Tipicamente la AOP più moderna offre linguaggi di programmazione *goal driven* basati sul modello di agente Belief-Desire-Intention (BDI)
  - Programmazione in stile *dichiarativo*
  - Semantica basata su una logica modale e un relativo sistema di ragionamento automatico
  - *Jason (jason.sourceforge.net)* è tra i più conosciuti

DSL per la AOP

- I sistemi distribuiti basati su agenti possono essere considerati uno specifico dominio applicativo
  - Sono caratterizzati da astrazioni di alto livello
  - Una volta scelta la tecnologia, le astrazioni offerte sono chiare e ben definite
  - Spesso vengono usati come strumenti per mettere in atto *specifiche eseguibili* di processi
- È quindi utile avere a disposizione DSL specifici
  - Sempre linguaggi di programmazione general purpose
  - Sempre legati a uno specifico modello di sistema multi-agente
  - Nel caso di JADE, che devono offrire supporto diretto alle astrazioni specifiche di JADE

JADEL

- JADE Language (JADEL) è un DSL per il dominio applicativo dei sistemi distribuiti realizzati con JADE
  - È un linguaggio con caratteristiche tipiche dei linguaggi di scripting
  - Offre supporto diretto alle astrazioni di JADE
  - Semplifica (in alcuni casi *notevolmente*) la realizzazione di sistemi che usano JADE
- JADEL è progettato sfruttando Xtext ed è strutturato su tre livelli distinti
  - *Astrazioni agent-oriented*: le cinque fornite da JADE
  - *Funzioni Xtend*: dialetto di Java basato su Xtext e disponibile con Xtext
  - *Espressioni Xbase* (tramite le funzioni Xtend): linguaggio per espressioni basato su Xtext e disponibile con Xtend

JADE per il Programmatore JADEL

```
ontology MeetingSchedulerOntology {
  concept Person(
    string name,
    aid AID,
    aid DFName)

  concept Appointment(
    aid inviter,
    string description,
    date startingOn,
    date endingWith,
    date fixedDate,
    many Person invited,
    many date possibleDates)
}
```

```
cyclic behaviour CancelAppointment
  for MeetingScheduler {
  on message msg
  when {
    performative is CANCEL
  } do {
    extract c as Appointment

    if(c.inviter == theAgent.AID)
      theAgent.cancelAppointment(
        c.fixedDate)
    else
      theAgent.removeAppointment(c)
  }
}
```

Conclusioni (I)

- Con il veicolo della MDE, i DSL stanno entrando in molti processi di ingegneria del software
  - Ma la tecnologia non è ancora matura, anche se molto sviluppata
- Nascono quindi questioni importanti da affrontare
  - Quando è conveniente introdurre un (nuovo) DSL in un processo?
  - Come si può quantificare il costo dell'introduzione di un (nuovo) DSL in un processo?
  - Per quali domini applicativi i DSL risultano vantaggiosi? Sotto quali condizioni specifiche?
  - Su quali strumenti è conveninente basare l'introduzione di un nuovo DSL? Basta XML o serve altro?
  - …

Conclusioni (II)

- La tecnologia degli agenti software sta entrando (*lentamente*) nel mainstream dello sviluppo dei sistemi distribuiti*
- L'utilizzo di Java ha agevolato l'adozione di JADE, ma la richiesta di un linguaggio di più alto livello è sempre più pressante da parte degli utenti
  - La sintassi di Java è molto tradizionale e non offre supporto ad astrazioni di più alto livello
  - JADE è sempre più soffisticato e quindi complesso da affrontare, almeno per chi inizia
  - Non è ancora disponibile un compromesso accettato tra la AOP dichiarativa e imperativa

* F. Bergenti, G. Caire, and D. Gotta. Large-scale network and service management with WANTS. *Industrial Agents: Emerging Applications of Software Agents in Industry*, Elsevier, 2015

AILAB

Artificial Intelligence Laboratory

Domain Specific Language per l'Ingegneria dei Sistemi Distribuiti

Prof. Federico Bergenti
federico.bergenti@unipr.it

www.ailab.unipr.it

Artificial Intelligence Laboratory
Università degli Studi di Parma
Parco Area delle Scienze 53/A
43124 Parma (Italy)

ailab@unipr.it