



UNIVERSIDADE DE SÃO PAULO

---

Escola de Artes, Ciências e Humanidades

Felipe Lombardi Pierin

**Desenvolvimento de um tradutor WSML para PDDL no  
contexto de um sistema de composição automática de  
serviços web**

São Paulo

Novembro de 2011

Universidade de São Paulo

Escola de Artes, Ciências e Humanidades

Felipe Lombardi Pierin

**Desenvolvimento de um tradutor WSML para PDDL no  
contexto de um sistema de composição automática de  
serviços web**

Monografia apresentada à Escola de Artes, Ciências e Humanidades, da Universidade de São Paulo, como parte dos requisitos exigidos na disciplina ACH 2018 – Projeto Supervisionado ou de Graduação II, do curso de Bacharelado em Sistemas de Informação.

**Modalidade:**

**TCC longo (anual) – monografia final**

São Paulo

Novembro de 2011

Universidade de São Paulo  
Escola de Artes, Ciências e Humanidades

Felipe Lombardi Pierin

**Desenvolvimento de um tradutor WSML para PDDL no  
contexto de um sistema de composição automática de serviços *web***

**Orientação**

---

Prof. Dr. Esteban Fernandez Tuesta

**Banca Examinadora:**

---

Profa. Dra. Fátima de Lourdes dos Santos Nunes Marques

---

Prof. Dr. Fábio Nakano.

São Paulo, Novembro de 2011

## **Agradecimentos**

Agradeço a Deus por me iluminar o caminho certo perante escolhas difíceis.

Agradeço a minha família que me apoiou incondicionalmente e que por vezes privou-se de interesses próprios para permitir com que eu pudesse continuar seguindo em frente com minha graduação.

Agradeço novamente a minha família pelos valores transmitidos e pela paciência comigo, não somente ao longo dos anos acadêmicos, mas ao longo da vida.

Agradeço aos meus orientadores José de Jesús Pérez Alcázar e Esteban Fernandes Tuesta por acreditarem e estarem sempre presentes para ajudar no bom andamento deste trabalho.

Agradeço aos meus (ex) professores que, ao seu modo, ensinaram no mais alto-nível utilizando criatividade, conhecimento e imposição de desafios.

Agradeço aos meus colegas com os quais tive a oportunidade de compartilhar dificuldades, incentivos, sorrisos e vitórias.

Agradeço a todos aqueles que colaboraram para a construção do meu caráter, para a minha graduação e em especial para a conclusão deste trabalho.

## Glossário

**ADL:** *Action Description Language.*

**AST:** *Abstract Syntax Tree.*

**ANTLR:** *ANother Tool for Language Recognition.*

**IA:** *Inteligência Artificial.*

**IDE:** *Integrated Development Environment.*

**PDDL:** *Planning Domain Definition Language.*

**RDF:** *Resource Description Framework.*

**STRIPS:** *Stanford Research Institute Problem Solver.*

**XML:** *Extensible Markup Language.*

**WSML:** *Web Service Modeling Language.*

**WSMO:** *Web Service Modeling Ontology.*

**YACC:** *Yet Another Compiler Compiler.*

## Resumo

Ao passo em que os serviços *web* promovem a interoperabilidade entre as organizações e, por essa razão, estão sendo cada vez mais adotados, o reuso de serviços *web* já existentes torna-se cada vez mais evidente. Ao reaproveitar um serviço *web* já construído, além de evitar o retrabalho, a tarefa de planejar e desenvolver uma determinada funcionalidade fica restrita a escolha ou a composição manual de uma ou mais funções que atinjam certo objetivo. Em algumas situações, o uso dos *webservices* publicados não é suficiente senão pela composição desses e, nesse sentido, como a composição de *web services* de forma manual pode se tornar uma tarefa demasiadamente onerosa, a agregação automática e semi-automática por meio da aplicação de inteligência artificial é almejado. Com o interesse de atender esse objetivo, a *web* semântica empregando definições ontológicas para descrever *web services* pode ser traduzida em linguagem de planejadores em inteligência artificial e, a partir dessa tradução, torna possível a tarefa de composição automática de *web services*. Nesse contexto, este trabalho de conclusão de curso focaliza o estudo e desenvolvimento de uma ferramenta para a tradução do domínio de linguagem WSML das definições ontológicas provindas de descrições oferecidas por *web services* que suportam WSMO para a linguagem de planejador de inteligência artificial conhecida por PDDL.

Palavras chaves: Linguagem de domínio PDDL para planejadores em inteligência artificial, *webservices* semânticos WSMO transcritos em WSML, tradutores de domínios de *web services* semânticos, composição de *web services* a partir da aplicação de inteligência artificial, tradução de ontologias em modelos de planejamento em inteligência artificial.

## Lista de Figuras

Figura 1 - Arquitetura do sistema de composição de serviços <i>web</i> .....	2
Figura 2 - Diagrama do processo de criação do sistema estocástico .....	3
Figura 3 – Exemplo de um relacionamento de dados .....	6
Figura 4 – Elementos base de um WSMO .....	7
Figura 5 – Definição de ontologia .....	8
Figura 6 – Variantes e camadas da linguagem WSML .....	9
Figura 7 – Estrutura de um problema de planejamento PDDL .....	11
Figura 8 – Exemplo de um domínio PDDL .....	12
Figura 9 - Exemplo de ação de domínio .....	12
Figura 10 – Exemplo de um problema PDDL .....	12
Figura 11 – Esquema do processo de compilação .....	13
Figura 12 – Fases da compilação .....	14
Figura 13 – Fluxo de dados de uma tradução com ANTLR .....	16
Figura 14 – Reconhecimento do fluxo de dados de entrada .....	16
Figura 15 – Arquitetura do padrão de projeto <i>Interpreter</i> .....	18
Figura 16 – Fases da elaboração do sistema de tradução .....	19
Figura 17 – Integração do Eclipse com o Antlr IDE .....	20
Figura 18 – Modelo do sistema de tradução de linguagens .....	21
Figura 19 - Interpretação do Antlr IDE sobre a gramática com definição de análise léxica de uma ontologia e de um <i>goal</i> .....	22
Figura 20 – Árvore sintática produzida pela interpretação do Antlr sobre o <i>goal</i> .....	23

Figura 21 – <i>Goal</i> WSML especificando uma viagem .....	23
Figura 22 – Ontologia de uma viagem em WSML.....	24
Figura 23 – Tradução de um objetivo WSML para um problema PDDL .....	25
Figura 24 – Tradução de uma ontologia WSML para um domínio PDDL .....	25
Figura 25 – Reconhecimento do Antlr sobre o PDDL gerado para uma ontologia .....	26
Figura 26 – Reconhecimento do Antlr sobre o PDDL gerado para um <i>goal</i> .....	26



## Sumário

1	Introdução .....	1
2	Objetivos .....	4
2.1	Objetivo Geral .....	4
2.2	Objetivos Específicos .....	4
3	Revisão Bibliográfica.....	5
3.1	Web Semântica.....	5
3.1.1	Web Services Semânticos .....	6
3.1.2	O Modelo conceitual WSMO.....	7
3.1.3	A linguagem WSML .....	9
3.2	O problema do planejamento.....	10
3.2.1	A linguagem PDDL.....	10
3.3	Compiladores .....	12
3.3.1	O compilador de compiladores ANTLR .....	15
3.4	Padrões de projeto.....	17
3.4.1	O padrão <i>Interpreter</i> .....	17
4	Metodologia .....	19
5	Resultados .....	22
6	Discussão .....	27
7	Conclusão e diretrizes para trabalhos futuros .....	29
8	Referências Bibliográficas .....	31

9	APÊNDICE A – Gramática da linguagem WSML.....	34
10	APÊNDICE B – Gramática do tradutor PDDL .....	37
11	APÊNDICE C– Árvore sintática de uma ontologia.....	41
12	APÊNDICE D – Árvore sintática de um <i>goal</i> .....	43
13	APÊNDICE E – Gramática de verificação do PDDL.....	44
14	ANEXO A – Exemplo de ontologia em WSML .....	47
15	ANEXO B – Exemplo de <i>goal</i> descrito em WMSL .....	48

# 1 Introdução

Em meio à globalização e a demasiada necessidade de sinergia entre diferentes organizações, os serviços *web*, conhecidos também por *webservices*, estão sendo cada vez mais utilizados a fim de suprir a demanda de troca de informações perante pontos geograficamente distribuídos. No entanto, ao passo em que as integrações são realizadas, mais *webservices* são construídos e isso leva, em algumas ocasiões, a reimplementação de sistemas já publicados e, conseqüentemente, em desperdício de tempo e de dinheiro.

A reimplementação de *webservices* é uma tarefa muito onerosa. Traduz-se no investimento em criação de algo já pronto e passível de interação. Nesse sentido, a capacidade de busca, composição e reutilização de serviços *web* já disponíveis é capaz de elevar a vantagem concorrencial de uma empresa que detém esse *know-how* perante outra que não o possui ao modo em que torna possível o enfoque sempre em novas funcionalidades nunca oferecidas por outra.

A reutilização de serviços *web* já disponíveis, por outro lado, não é uma tarefa simples e pode tornar-se extremamente cara. O dinamismo e o nascimento ininterrupto de novos serviços sem descrição na internet faz com que uma busca manual de um serviço que atenda uma determinada demanda específica possa ser praticamente impossível. A aplicação de agentes de software e da *web* semântica torna isso possível e automático.

A *web* semântica é um modelo que agrega significado aos serviços disponibilizados na internet. Esse modelo, ao ser utilizado nos *webservices*, permite a interpretação de sistemas por meio agentes de software que, ao seu modo, podem ser transcritos em uma linguagem de planejamento que permitem a aplicação de inteligência artificial (IA). Valendo-se desse contexto, os pesquisadores Digiampietri, Pérez-Alcázar e Medeiros (2007) arquitetaram um mecanismo, conforme ilustrado na figura 1, capaz de aplicar técnicas de IA em *webservices* semânticos para tornar possível a composição automática de serviços *web*.

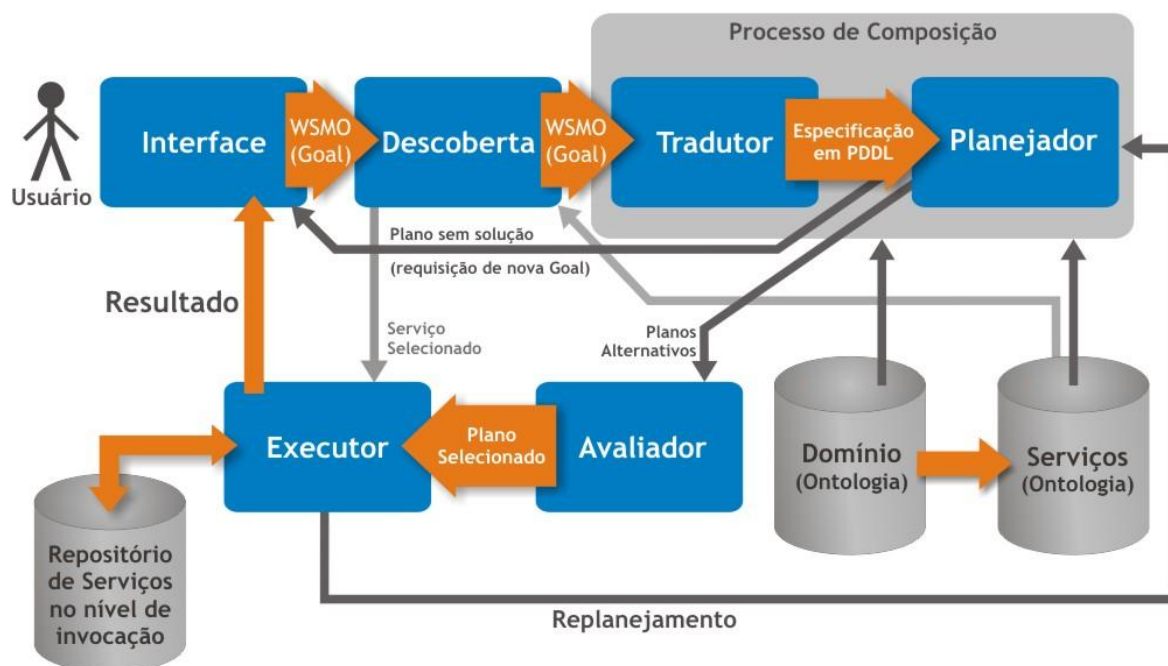


Figura 1 - Arquitetura do sistema de composição de serviços *web* (PÉREZ-ALCÁZAR, 2007)

A arquitetura do sistema de composição de serviços *web* compreende cinco módulos. A “Interface” é responsável por receber requisições de linguagem WSML que descrevem *webservices* semânticos do tipo WSMO e enviá-las ao tradutor na forma de *goals*. A “Descoberta” encontra um serviço publicado adequado a uma meta dentro do repositório de serviços. O “Tradutor” é capaz de mapear uma requisição de domínio WSML em um domínio de planejamento do tipo PDDL. O “Planejador” gera uma política ou plano de solução a partir de uma solicitação PDDL e o “Executor” que dado um plano de solução, realiza um mapeamento em serviço WSML equivalente.

Neste trabalho, será realizado o estudo e desenvolvimento do módulo “Tradutor” que, de acordo com Pérez-Alcázar (2007), apresenta duas funcionalidades fundamentais. A tradução de domínios e metas WSML em especificação de problema PDDL e alimentação do repositório de ontologias de domínio e do repositório de ontologias de serviços como mostrado na figura 2.

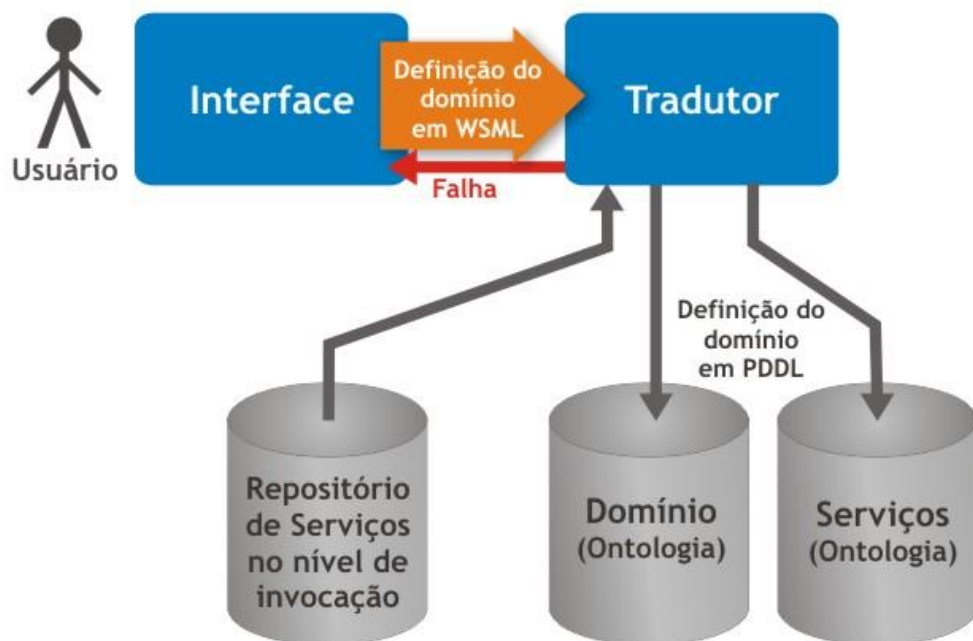


Figura 2 - Diagrama do processo de criação do sistema estocástico (PÉREZ-ALCÁZAR, 2007)

## **2 Objetivos**

### **2.1 Objetivo Geral**

O objetivo geral deste trabalho consistiu na pesquisa e elaboração de um arcabouço capaz de tornar viável a tradução metas e ontologias especificadas em WSML para a linguagem de planejamento PDDL no contexto de um sistema de composição automática de serviços *web*.

### **2.2 Objetivos Específicos**

Os objetivos específicos deste trabalho foram:

- A aplicação de compiladores para tradução de linguagens;
- O desenvolvimento de um analisador léxico WSML;
- O desenvolvimento de um analisador sintático WSML;
- O desenvolvimento de um analisador semântico WSML;
- O desenvolvimento de um sistema tradutor de linguagem WSML para linguagem PDDL;

## 3 Revisão Bibliográfica

### 3.1 Web Semântica

O significado dos dados no atual contexto dos recursos de informação é muitas vezes ausente e demanda uma programação complexa ou um usuário para supri-lo (HEBELER et al., 2009). Nesse cenário, o estabelecimento de contextos ou significados aderentes a uma gramática ou às construções de determinada linguagem, poderia tornar eficiente o uso dos dados subjacentes, tornando possível a descrição e a correlação da *web* de dados (HEBELER et al., 2009).

A semântica permite com que um símbolo tenha um significado a partir de relacionamentos (HEBELER et al., 2009). Ao passo em que existindo significados e relações bem definidos é fácil para o leitor humano interpretá-los, para que a máquina chegue numa mesma conclusão partindo de raciocínio lógico é necessária a codificação do entendimento humano para linguagem de máquina, afinal a máquina não ganha entendimento real (FENSEL et al., 2007). Essa codificação do entendimento humano sobre dados a fim de permitir a interpretação através de agentes de *software* é o âmago da *web* semântica.

A *web* semântica foi o termo escolhido por Tim Berners-Lee<sup>1</sup> (FENSEL et al., 2007) para detalhar a geração em que os computadores seriam providos de inteligência de leitura originada através de vocabulários interligados usados pelos autores da *web* para definir explicitamente as palavras e conceitos deles. Isso permitiria a agentes de software realizar inferências inteligentes que vão além de simples análise lingüística providas pelas atuais máquinas de busca (ALESSO; SMITH, 2005).

As ligações entre recursos de informação e terminologias formais são expressas por anotações chamadas ontologias (FENSEL et al., 2007). Elas permitem às máquinas compreender a informação através das ligações entre os recursos da informação e os termos

---

<sup>1</sup> BERNERS-LEE, T.; FISCHETTI, M. **Weaving the Web**: The past, present and future of the World Wide Web by its inventor, 1999, 272p.

da ontologia o que facilita a interoperabilidade pois são uma especificação formal explícita de uma conceituação compartilhada. (FENSEL et al., 2007).

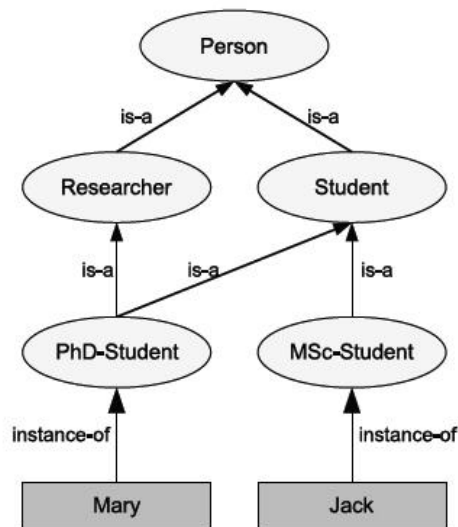


Figura 3 – Exemplo de um relacionamento de dados (FENSEL et al., 2007)

### 3.1.1 Web Services Semânticos

Um *webservice* é uma aplicação de negócio aberta, orientada a internet, modular e baseada em interfaces padrões que pode ser identificada por um URI na qual suas interfaces e ligações são capazes de serem definidas, descritas e descobertas como artefatos XML (ALONSO et al., 2004). São sistemas que permitem a existência e criação de *softwares* clientes que se ligam e interagem em aplicações mais complexas e distribuídas (ALONSO et al., 2004). Já os *webservices* semânticos são o próximo estágio dos atuais *web services* (FENSEL et al., 2007) e constituem-se na aplicação da *web* semântica sobre os serviços *web* com foco na automatização de todos os estágios da vida destes (ALONSO et al., 2004).

A automatização de todos os estágios da vida um serviço *web* se dá a partir da padronização da representação e manipulação dos metadados semânticos usados para descrever a eles e a todos os aspectos usados neles (ALONSO et al., 2004). Essa padronização é importante porque permite o desenvolvimento de ferramentas que automatizam o uso de serviços *web*, especialmente na descoberta de *web services* semanticamente equivalentes, ou



seja, a padronização permite ir além da busca de *webservices* que suportam uma interface ou um protocolo conhecido (ALONSO et al., 2004).

Segundo Alonso et al. (2004), existem muitos trabalhos evoluindo no sentido de automatizar o uso de serviços *web* semânticos e que giram em torno de especificações como a *RDF and RDF Schema*, a *DAML\_OIL*, a *OWL*, a *DAML-S* e mais recentemente o *WSMO*.

### 3.1.2 O Modelo conceitual WSMO

De acordo com Fensel, Kerrigan e Zaremba (2007), o *Web Service Modeling Ontology* (WSMO) é um modelo conceitual que descreve todos os aspectos relacionados a serviços *web* semânticos e que torna possível a integração entre estes. É uma meta-ontologia criada para descrever todos os aspectos de *web services* semânticos (KASHYAP; BUSSLER; MORAN, 2008) e que consiste de quatro elementos bases: *web services*, objetivos (*goal*), ontologias (*ontology*) e mediadores (*mediator*) conforme a figura 4 ilustra.

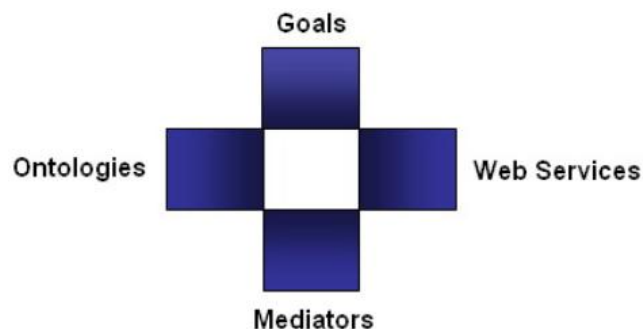


Figura 4 – Elementos base de um WSMO (FENSEL et al., 2011)

De acordo com Fensel et al. (2011), o sucesso de um serviço *web* semântico é proveniente da existência de ontologias e portanto não é de se impressionar que ela seja descrita como um dos quatro elementos base do modelo WSMO, afinal, é dela a responsabilidade de conectar as descrições de todos os elementos do modelo a partir da agregação de significado. Elas são geralmente compostas de anotações simples, ontologias importadas de outras ontologias, mediadores responsáveis por mediar diferenças ontológicas, conceitos que representam a ligação entre terminologia e domínios, relações e funções que determinam a conexão entre

conceitos, instâncias que permitem herança e por fim axiomas que provêem um mecanismo de se adicionar expressões lógicas arbitrárias a ontologia conforme representa a figura 5.

```
Class ontology
  hasAnnotations type annotations
  importsOntology type ontology
  usesMediator type ooMediator
  hasConcept type concept
  hasRelation type relation
  hasFunction type function
  hasInstance type instance
  hasRelationInstance type relationInstance
  hasAxiom type axiom
```

Figura 5 – Definição de ontologia (FENSEL et al., 2011)

As entidades computacionais que provêem alguma funcionalidade que possuem um valor associado em um determinado domínio são chamadas de *webservices* (FENSEL et al., 2011). Deles são extraídos todos os dados referentes em torno de capacidades e interfaces de comunicação. Eles são compostos de propriedades não funcionais que agem como anotações simples, capacidades que, compostas de vários axiomas, descrevem precondições, resultados e efeitos de um serviço web e por fim as interfaces que representam a forma com que uma capacidade pode ser atingida.

Os desejos ou objetivos do usuário são outro importante alicerce do modelo WSMO especificado pelos *Goals* (FENSEL et al., 2011). Eles descrevem os aspectos relacionados aos desejos do usuário com respeito as funcionalidades e comportamentos requeridos de um serviço usando ontologias como especificação formal. Nas palavras de Fensel, Kerrigan e Zaremba (2007), o *goal* modela a visão do usuário com relação ao uso de um serviço *web*.

A interoperabilidade no nível de dados, processos ou protocolos de serviços ou ontologias é uma das questões centrais do modelo que contempla os *goals* e *ontologies*. Conforme afirma Bruijn (2008), a interoperabilidade é um importante obstáculo a ser superado para permitir larga abrangência entre serviços semânticos. Nesse contexto, a especificação detalhada de como a interação entre objetos acontece é justamente o papel do elemento base chamado mediador (ou *mediator*) que assim como todos os outros elementos do modelo WSMO possuem anotações e podem importar ontologias.

### 3.1.3 A linguagem WSML

O *Web Service Modeling Language* (WSML) é uma linguagem formal concreta para a descrição de ontologias e *web services* semânticos que leva em consideração todos os aspectos de descrição de *web services* identificados pelo WSMO (BRUIJN et al., 2008). É uma linguagem que por ser voltada a *internet* e adota padrões especificados pela W3C como o XML, o RDF e IRI que identificam objetos e recursos (FENSEL et al., 2011).

O WSML reconhece duas camadas ou paradigmas importantes: lógicas de descrição e lógicas baseadas em regras de linguagem (BRUIJN et al., 2008). Essa característica possibilita ao usuário escolher qual paradigma usar através de diferentes variantes (WSML-Core, WSML-DL, WSML-Flight, WSML-Full) que diferem em expressividade lógica e complexidade assim como mostrado na figura 6.

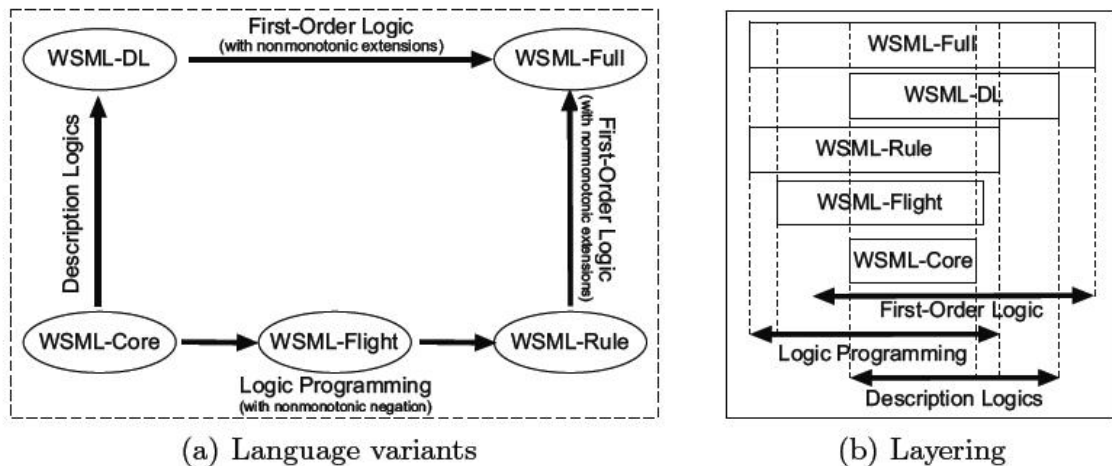


Figura 6 – Variantes e camadas da linguagem WSML (FENSEL et al., 2007)

Entre as variantes de uma linguagem WSML, é o WSML-Core o menos expressivo. Ele possui conceitos, atributos, instâncias, relações binárias, relações hierárquicas e suporte a tipos. O WSML-DL tem uma relação com outra linguagem para *webservices* semânticos, o OWL. Por sua vez, o WSML-Flight é uma extensão do WML-Core que acrescenta recursos como metamodelo, restrições e negação não monotônica. Já o WSML-Rule é uma extensão do WSML-Flight com ainda mais recursos. Finalmente, o WSML-Full é a linguagem que unifica os três modelos citados anteriormente.

## 3.2 O problema do planejamento

Ao passo em que o planejamento é definido para Russel e Norvig (2004) como a tarefa de apresentar uma sequência de ações que alcançarão um objetivo conhecido, para Ghallab, Nau e Traverso (2004) é um processo que escolhe e organiza ações, antecipa resultados e procura atingi-los da melhor forma possível. Ainda para Ghallab, Nau e Traverso (2004) pode também ser considerado uma área da IA que estuda esse problema computacionalmente.

Conforme Ghallab, Nau e Traverso (2004), muitas de nossas ações cotidianas não requerem planejamento e isso decorre do fato de que quando o propósito da ação é imediato ou quando reproduzimos comportamentos bem treinados ou quando o curso da ação pode ser livremente adaptado conforme a ação acontece, geralmente agimos e adaptamos sem explicitamente planejá-las. Por outro lado, quando uma ação demanda um planejamento, ela é geralmente restrita a um ambiente crítico e envolve alto risco ou alto custo. Acontece que o processo de planejar pode ser uma tarefa complicada que demanda tempo e dinheiro e nesse sentido pode não ser passível de aplicação constante.

A fim de permitir o acesso aos recursos de planejamento e, principalmente, pagáveis é que surgem ferramentas de processamento de informação visando a automação do planejamento (GHALLAB; NAU; TRAVERSO, 2004). Assim surgiram representações de ações como a linguagem STRIPS, a *Action Description Language* (ADL) que relaxa algumas das restrições da linguagem STRIPS tornando possível codificar problemas mais realistas e por fim a PDDL que por sua vez é desde 1998 a linguagem padrão para competidores na conferência AIPS (*Artificial Intelligence Planning Systems*) porque é uma sintaxe padronizada e analisável por computador para representar STRIPS, ADL e outras linguagens (RUSSEL; NORVIG, 2004).

### 3.2.1 A linguagem PDDL

Originalmente desenvolvida pelo comitê da competição do AIPS-98 para ser usada na definição de problemas de planejamento (GHALLAB et al., 1998), a linguagem PDDL (*Planning Domain Definition Language*) é hoje a codificação padrão para tarefas de planejamento clássico (RUSSEL; NORVIG, 2004). Isso significa que ela pode atuar sobre

ambiente completamente observável, determinístico, finito, estático (quando uma mudança só ocorre face à ação de um agente) e discreto, ou seja, com a existência de tempo, ação, objetos e efeitos.

A descrição de um problema de planejamento a partir da linguagem PDDL acontece a partir de duas entradas ilustradas conforme a figura 7: a definição do domínio e a definição do problema. Esses recursos respeitam a neutralidade característica da linguagem que permite a ela não adaptar-se melhor a algum planejador em particular o que torna possível a realização de comparações justas entre diferentes planejadores. Em outras palavras, ela expressa a física de um domínio, definindo predicados, ações, estrutura que compõem as ações e os efeitos delas (GHALLAB et al., 1998) tornando viável aos planejadores encontrar um determinado objetivo sem a necessidade da existência de dicas de como fazê-lo, algo muito comum na maioria dos planejadores da época.

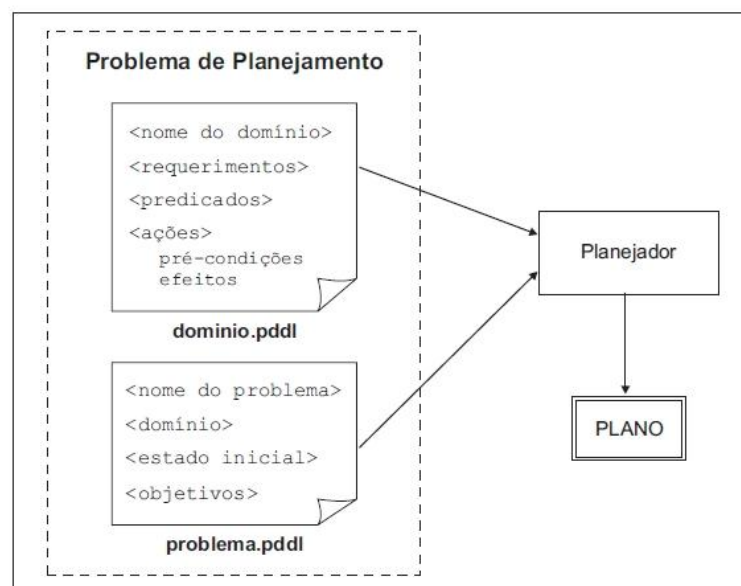


Figura 7 – Estrutura de um problema de planejamento PDDL (BENACCHIO, 2008)

Um domínio em linguagem PDDL é composto de requerimentos (*requirements*), tipos (*types*), constantes (*constants*) e predicados (*predicates*) conforme se observa diante da figura 8. Os requerimentos são os subconjuntos de funcionalidades que um domínio demanda de suporte por parte dos planejadores. A importância dos requerimentos são, de acordo com Ghallab et al. (1998), uma forma de permitir com que um planejador descarte rapidamente um domínio que ele não seja capaz de atender.

```

(define (domain briefcase-world)
  (:requirements :strips :equality :typing :conditional-effects)
  (:types location physob)
  (:constants (B - physob))
  (:predicates (at ?x - physob ?l - location)
               (in ?x ?y - physob))
  ...

```

Figura 8 – Exemplo de um domínio PDDL (GHALLAB et al., 1998)

Um domínio em linguagem PDDL é também composto de ações (*action*) que respeitam parâmetros, pré-condições e efeitos. Tomando como base a representação ilustrada na figura 9, observa-se a descrição da ação de movimentar um objeto de uma localização (?m) até outra (?l) e entende-se também que a posição final não pode ser a posição inicial ainda que o efeito dela é posicionar um objeto em uma nova posição.

```

(:action mov-b
  :parameters (?m ?l - location)
  :precondition (and (at B ?m) (not (= ?m ?l)))
  :effect (and (at b ?l) (not (at B ?m))
              (forall (?z)
                (when (and (in ?z) (not (= ?z B)))
                  (and (at ?z ?l) (not (at ?z ?m)))))) )

```

Figura 9 - Exemplo de ação de domínio (GHALLAB et al., 1998)

A definição do objetivo a ser atingido por um planejador é descrito no contexto da linguagem PDDL através do recurso problema (Figura 10). O problema é a definição do que o planejador deve resolver e é baseada de acordo com um domínio. Nele há a especificação de como se encontra o ambiente dada por condições verdadeiras estabelecidas na seção “*init*” e o objetivo que se espera alcançar (na seção “*goal*”).

```

(define (problem get-paid)
  (:domain briefcase-world)
  (:init (place home) (place office)
         (object p) (object d) (object b)
         (at B home) (at P home) (at D home) (in P))
  (:goal (and (at B office) (at D office) (at P home))))

```

Figura 10 – Exemplo de um problema PDDL (GHALLAB et al. 1998)

### 3.3 Compiladores

Os sistemas de computador que são capazes de traduzir um programa escrito em uma linguagem-fonte para uma linguagem-alvo equivalente são chamados de compiladores (AHO et al., 2007; COOPER; TORCZON, 2004; LOUDEN, 2004). Esses sistemas são programas

geralmente complexos e grandes, chegando a somar uma quantidade significativa de linhas de código (COOPER; TORCZON, 2004; LOUDEN, 2004). Isso torna a tarefa de escrever um sistema com tais características algo não trivial e, por isso, a maioria dos cientistas e profissionais de computação jamais escreverão um compilador completo (LOUDEN, 2004).



Figura 11 – Esquema do processo de compilação (LOUDEN, 2004)

A construção de um compilador é um trabalho que pode ser dividido levando em consideração as diferentes fases, conforme se observa com a figura 12, que compõe esse tipo de programa entre elas a análise léxica, a análise sintática, a análise semântica, geração de código e, por fim, a otimização do código gerado. A análise léxica, também conhecida por sistema de varredura, acontece conforme a sequência de caracteres do programa fonte é lida. Durante a varredura, as sequências de caracteres são organizadas como unidades significativas denominadas *tokens*, por exemplo, no inglês (LOUDEN, 2004). A análise sintática é o ciclo, geralmente representado como uma árvore sintática, que recebe do sistema de varredura o código fonte na forma de *tokens* e determina a estrutura do programa determinando os elementos estruturais e os relacionamentos entre eles (LOUDEN, 2004). O analisador semântico verifica o significado do código-fonte contrastando com a sintaxe dele (AHO et al., 2007). O otimizador de código fonte é o processo de melhoria do código fonte. O módulo gerador de código é o responsável por gerar a partir do código fonte já melhorado o código alvo e por fim o otimizador do código alvo tenta melhorar o código alvo gerado (AHO et al., 2007).

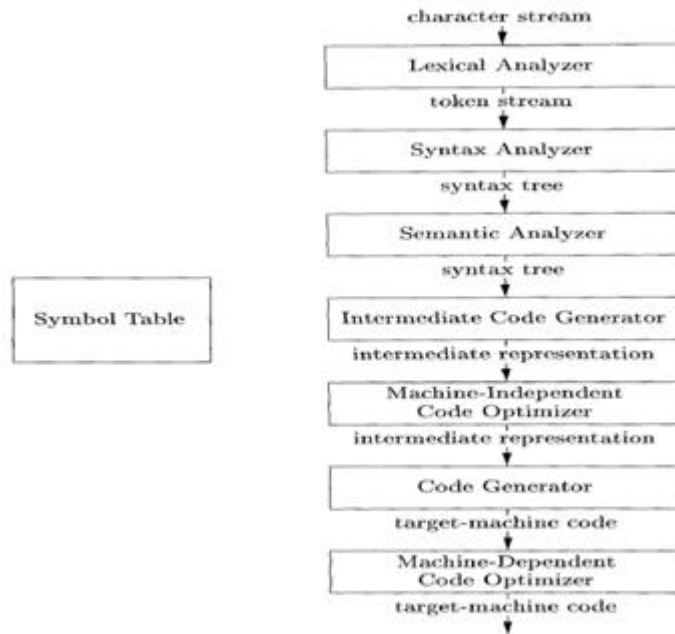


Figura 12 – Fases da compilação (AHO et al., 2007)

A crescente necessidade do desenvolvimento de compiladores incentivou a pesquisa na automatização de algumas das fases da compilação (SAFONOV, 2010). O estudo dos autômatos finitos auxiliou na criação de geradores de sistemas de varredura dos quais o LEX<sup>2</sup> é o mais conhecido (LOUDEN, 2004). O estudo de programas que automatizam apenas a fase da análise sintática, chamados de geradores de analisadores sintáticos, contempla o YACC (*Yet Another Compiler Compiler*) como ferramenta mais conhecida (LOUDEN, 2004). Em geral as pesquisas tinham como objetivo comum o desenvolvimento de um compilador de compiladores.

Um compilador de compiladores é uma ferramenta que usa uma definição formal de sintaxe e semântica de uma linguagem e gera um compilador daquela linguagem (SAFONOV, 2010). O modelo teórico de compilador de compiladores é baseado no formalismo inventado por Knuth chamado de gramática atribuída. Nele combina-se uma definição formal da sintaxe de uma linguagem de programação com a semântica, ou seja, ações com significados atribuídas a cada regra de sintaxe para avaliar as propriedades semânticas ligadas cada símbolo participando da regra sintática (SAFONOV, 2010). Algumas

<sup>2</sup> LABORATORIES, A.; LESK, M. **LEX**: A Lexical Analyzer Generator, 1975, 35 p.



ferramentas que suportam essa característica são o JavaCC, o CoCo/Re SableCC e o ANTLR (SAFONOV, 2010).

### 3.3.1 O compilador de compiladores ANTLR

O Antlr é um conjunto de ferramentas amplamente utilizado na indústria voltado para o desenvolvimento de compiladores modernos que contem geradores léxicos e sintáticos (SAFONOV, 2010). É considerado ótimo para aplicação em cenários em que se encaixam as tarefas de reconhecimento e tradução de linguagens (PARR, 2007). Algumas das aplicações que fazem uso dele são: o desenvolvimento em escala mundial do ambiente de desenvolvimento NetBeans e da ferramenta de persistência em banco de dados Hibernate (SAFONOV, 2010).

O compilador de compiladores ANTLR é versátil com relação à geração de código fonte. Ele é capaz de gerar código em linguagens largamente adotadas como são o Java, o C#, o Action Script e o Java Script (SAFONOV, 2010). Além do mais, é uma ferramenta orientada a objeto e portanto, todo o código gerado através dela é também um código orientado a objetos (SAFONOV, 2010).

A especificação da análise léxica e sintática no Antlr se dá a partir da gramática, um arquivo de entrada de extensão *grammar* (.g). A gramática contém a descrição de uma linguagem formal e permite ao compilador de compiladores de gerar um programa que determina se obedecem àquela linguagem sendo que na medida em que são adicionados trechos de código a essa gramática inicial, o reconhecedor se torna um tradutor (PARR, 2007).

Segundo Parr (2007), o tradutor do Antlr é um programa capaz de mapear cada sentença de entrada da linguagem em uma sentença de saída incorporando a propriedade de realizar diferentes ações para múltiplas sentenças. Essa característica é obtida operando-se sobre a cadeia de palavras que entram (análise léxica), realizando o *parsing* (análise sintática) sobre a cadeia de símbolos do vocabulário, chamados *tokens*, emanados do analisador léxico e, por fim, acrescentando ações sobre a gramática a partir da introdução de trechos de código únicos para os vários fragmentos da sentença conforme a figura 13.

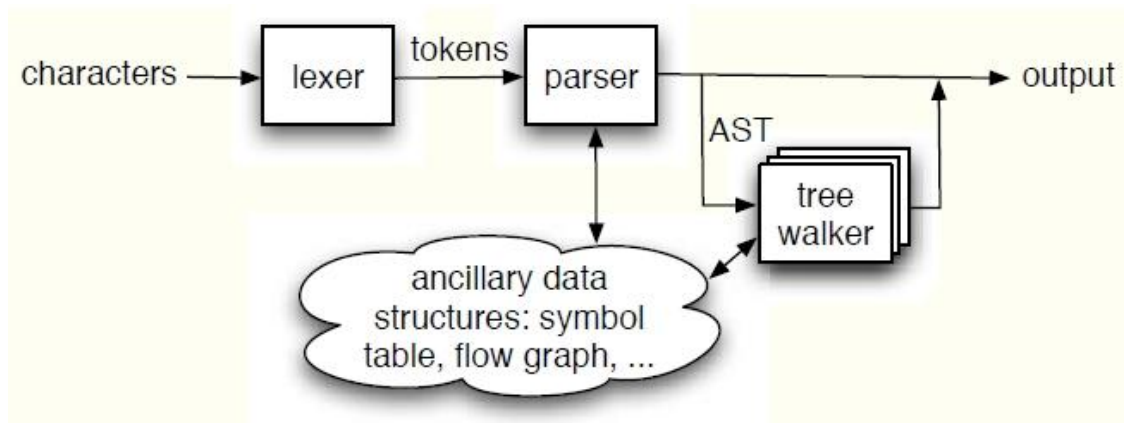


Figura 13 – Fluxo de dados de uma tradução com ANTLR (PARR, 2007)

O processo de reconhecimento do fluxo de dados no Antlr é otimizado conforme ilustra a figura 14. Conforme acontece a análise léxica, um objeto do tipo *Charstream* é alimentado com caracteres até que todos os caracteres sejam armazenados em memória. Nesse momento, os símbolos, ou *tokens*, pré-definidos vão apontando os caracteres do fluxo armazenado ao invés de criar novos objetos. De forma similar, as árvores sintáticas abstratas, ou *abstract syntax tree* (AST), vão sendo criadas apontando tokens. O *CommonTree* é um exemplo de objeto que armazena a posição de um símbolo na memória.

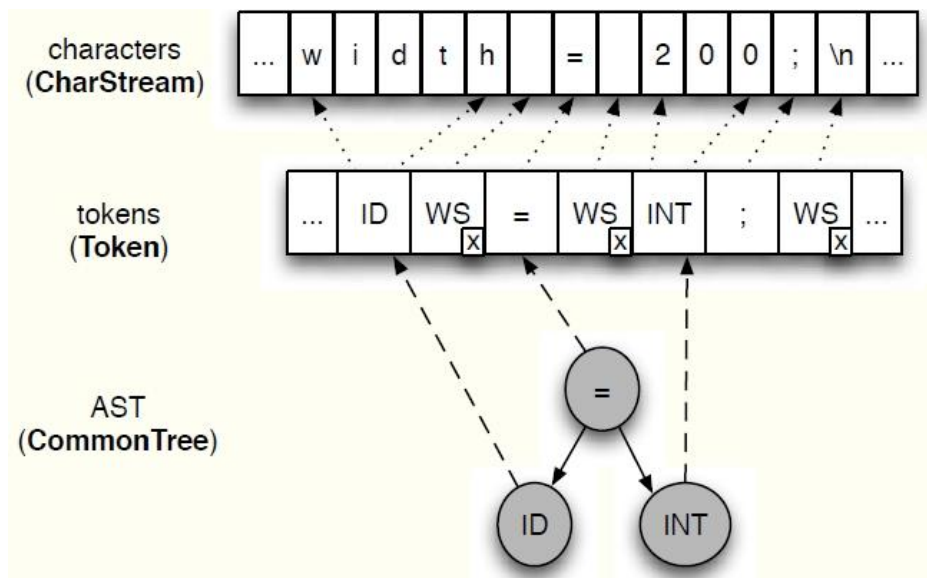


Figura 14 – Reconhecimento do fluxo de dados de entrada (PARR, 2007)

O relacionamento elucidado na imagem anterior ainda ilustra símbolos marcados que representam um mecanismo elegante do Antlr, os canais de *tokens*. Os canais de *tokens* são uma forma de ignorar e ao mesmo tempo não se desfazer de espaços em brancos ou comentários. Como o analisador sintático sintoniza apenas um canal ignorando os outros, aqueles símbolos marcados acabam armazenados em um canal escondido (*Hidden Channel*). No entanto, a ordem relativa dos caracteres de entrada é armazenado independentemente de canais existentes.

### 3.4 Padrões de projeto

Um padrão de projeto descreve um problema no nosso ambiente e o cerne de sua solução (GAMMA et al., 2004). Ele nomeia, abstrai e identifica aspectos-chaves de uma estrutura de projeto comum para torná-la útil para criação de um projeto orientado a objetos reutilizável. É normalmente composto de nome, problema, solução e conseqüências, de maneira a descrever em que situação pode ser aplicado, se ele pode ser aplicado em função de outras restrições de projeto e as conseqüências, custos e benefícios de sua utilização.

Os padrões de projeto, ou *design patterns*, são técnicas consideradas boas práticas de programação que auxiliam na resolução de problemas de arquitetura do paradigma de orientação a objetos. A aplicação desses padrões em projetos facilita o entendimento dos novos desenvolvedores ,tornado mais fácil de reutilizar projetos e arquiteturas bem sucedidas, uma vez que são expressões de técnicas testadas e aprovadas (GAMMA et al., 2004). Os padrões de projeto ainda podem melhorar a documentação e manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente.

#### 3.4.1 O padrão *Interpreter*

O padrão de projetos *Interpreter* é uma técnica bastante utilizada quando se há uma linguagem para interpretar e é possível representar sentenças dela como árvores sintáticas abstratas (GAMMA et al., 2004). Não por outro motivo, são amplamente utilizados em

compiladores de linguagens orientadas a objetos como Smalltalk (GAMMA et al., 2004). Isso é possível pois nele, dada uma linguagem, pode-se definir uma representação para a sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças daquela linguagem.

A forma de definição de uma gramática para linguagens simples, representação na linguagem e explicação dessas sentenças é descrita no *pattern Interpreter* conforme o diagrama de classes apresentado na figura 15. As regras da gramática são representadas por classes o que torna possível usar herança para mudar ou estendê-las. As classes definem os nós na árvore sintática abstrata e tem construções similares. Juntas, as características do padrão *Interpreter* facilitam o desenvolvimento permitindo a modificação incremental e a inclusão de novas expressões que podem ser definidas a partir de variações de velhas expressões.

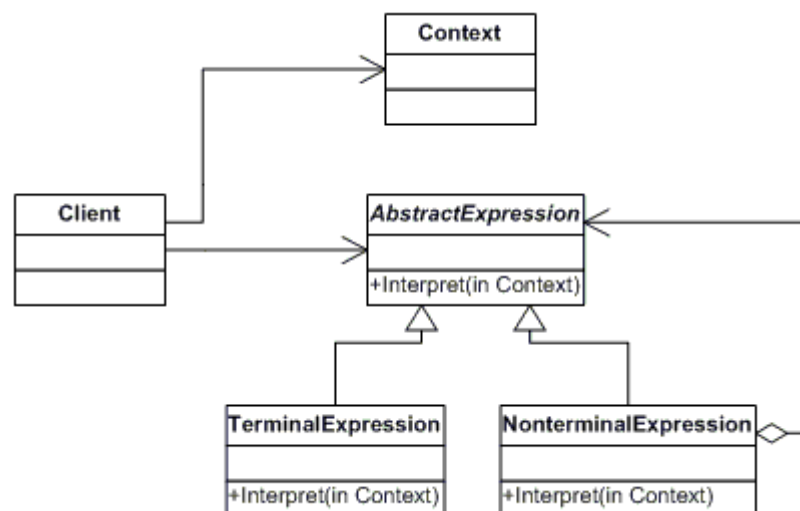


Figura 15 – Arquitetura do padrão de projeto *Interpreter* (GAMMA et al., 2004)

## 4 Metodologia

O desenvolvimento do compilador responsável pela tradução da linguagem WSMML para PDDL foi elaborado respeitando, como observado na figura 16, quatro fases principais: a definição do escopo, a definição das ferramentas para o desenvolvimento do sistema, o desenvolvimento do sistema propriamente dito e a validação do resultado.

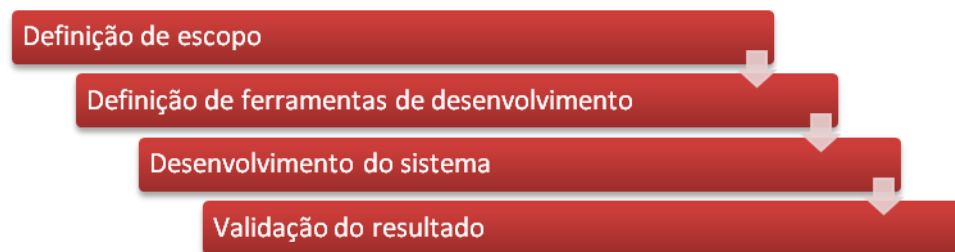


Figura 16 – Fases da elaboração do sistema de tradução

Na definição do escopo foram decididas a abrangência da tradução da linguagem WSMML e os recursos a serem traduzidos levando em consideração o contexto do sistema de composição automática de *webservices* do projeto descrito por Pérez-Alcázar (2008). Nesse momento, em face da complexidade da construção de um compilador e do tempo limitante para conclusão desta monografia, o escopo do sistema foi restringido à tradução das ontologias e *goals* descritos em WSMML. Os modelos de ontologias e objetivos do usuário, respectivamente o anexo A e o anexo B deste trabalho, foram escolhidos por serem exemplos propostos por Fensel et al. (2011) e que não requerem o uso de mediadores ou múltiplas extensões de ontologias que é justamente o interesse deste trabalho.

Na fase de definição de ferramentas de desenvolvimento foram escolhidos instrumentos para a elaboração do sistema de forma a estimular a continuação do projeto por outros indivíduos interessados. Nesse sentido, optou-se pela utilização do Java por ser uma linguagem de programação amplamente conhecida e de fácil acesso, a adoção do Eclipse como ambiente integrado de desenvolvimento (IDE) em virtude da existência de uma extensão que facilita a manipulação do compilador de compiladores Antlr, o ANTLR IDE e o Antlr como compilador de compiladores devido à existência de documentação, a capacidade de abranger várias fases de um compilador e facilidade de geração de códigos orientados a

objetos além da adaptabilidade à linguagem Java. Ainda nesse contexto, os códigos fonte tiveram a versão controlada com o gerenciador de versões GIT e disponibilizados para visualização ou participação colaborativa através do portal GitHub<sup>3</sup>.

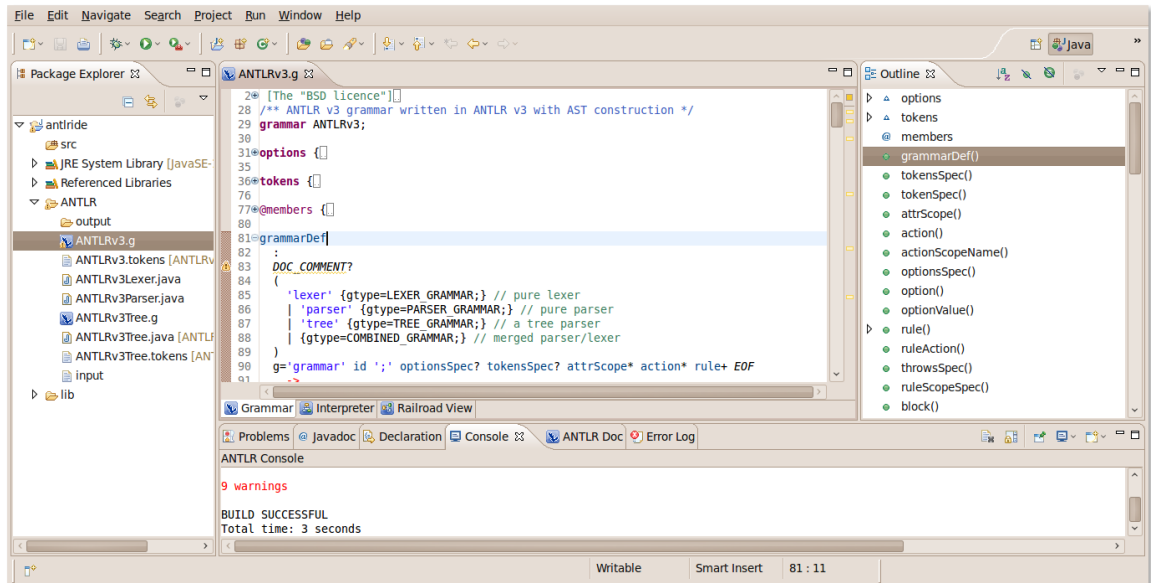


Figura 17 – Integração do Eclipse com o Antlr IDE (ANTLR IDE)

O desenvolvimento do programa capaz de traduzir as linguagens previamente definidas no escopo do projeto foi baseado no modelo de sistema apresentado na figura 18. Nesse momento, foram construídas as gramáticas de análise léxica e análise sintática do WSMML para a interpretação do Antlr e conseqüente geração de analisadores léxicos e sintáticos. Ao passo em que os analisadores léxicos e semânticos foram construídos, foi possível desenvolver a gramática, localizada no apêndice B, que agrega a informação de como realizar a tradução da linguagem de entrada na linguagem alvo. Nesta última linguagem, fez-se o uso do padrão de projetos *Interpreter* para lidar com as diferentes regras da gramática do WSMML e a forma pela qual elas deveriam ser traduzidas.

<sup>3</sup> Acessível em: <https://github.com/fpierin/wsml2pddl/>

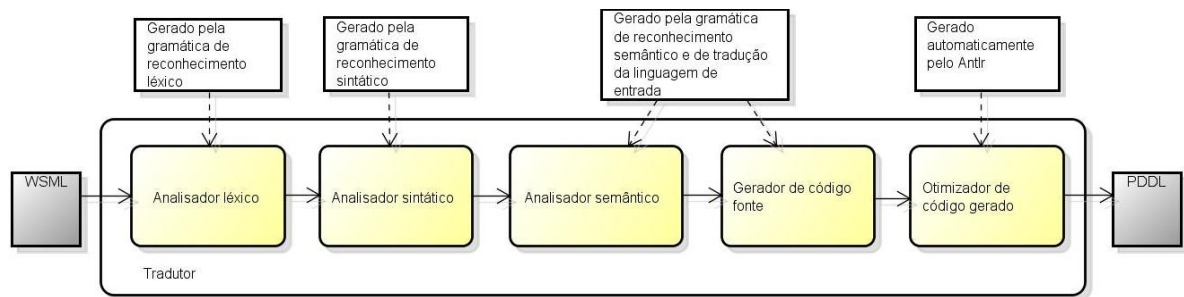


Figura 18 – Modelo do sistema de tradução de linguagens

Após a conclusão do desenvolvimento do sistema, os resultados obtidos através da tradução dos exemplos WSML propostos foram validados a partir do reconhecimento e conseqüente geração de árvores sintáticas equivalentes no AntlrIDE. A gramática do artefato PDDL usada para o reconhecimento do Antlr se baseou na sintaxe de PDDL proposta por Ghallab et al. (2008), do apêndice E deste trabalho.

## 5 Resultados

O desenvolvimento do sistema de tradução de ontologias e objetivos resultou na criação de um compilador com capacidade de análise léxica, sintática e semântica ilustrado no modelo de tradução de linguagens da Figura 18. As regras que definem o analisador léxico estão na gramática do apêndice A desenvolvida no decorrer deste trabalho e pode ser observada na figura 19 conforme a interpretação gráfica do Antrl IDE destacando as palavras chaves.

```
1 wsmmlVariant _http://www.wsmo.org/wsmml-syntax/wsmml-flight
2
3 namespace { _"http://www.gsomo.org/dip/travel/domainOntology#",
4   dc _"http://purl.org/dc/elements/1.1#",
5   wsmml _http://www.wsmo.org/wsmml-syntax# }
6
7 ontology TravelOntology
8 concept Ticket
9 annotations
10 dc#description hasValue "concept of a ticket"
11 endAnnotations
12 from ofType Region
13 to ofType Region
14 vehicle ofType Vehicle
15
16 concept Region
17
18 concept Country subConceptOf Region
19 name ofType _string
20
21 concept City subConceptOf Region
22 name ofType _string
23 country ofType Country
24
25 concept EUCity subConceptOf City
26
27 concept GermanCity subConceptOf EUCity
```

```
1 wsmmlVariant _http://www.wsmo.org/wsmml-syntax/wsmml-flight
2
3 namespace { _"http://www.gsomo.org/dip/travel/goal#",
4   dO _"http://www.gsomo.org/dip/travel/domainOntology#",
5   dc _"http://purl.org/dc/elements/1.1#" }
6
7 goal _http://www.gsomo.org/dip/travel/goal.wsmml
8
9 importsOntology _"http://www.gsomo.org/dip/travel/domainOntology#TravelOntology"
10
11 capability goalCapability
12 postcondition
13   definedBy
14     ?ticket [ dO#from hasValue ?from,
15               dO#to hasValue ?to ] memberOf dO#Ticket and
16     ?from memberOf dO#AustrianCity and
17     ?to memberOf dO#UKCity.
```

Figura 19 - Interpretação do Antrl IDE sobre a gramática com definição de análise léxica de uma ontologia e de um *goal*

A capacidade de análise sintática, assim como a da análise léxica, foi atingida em virtude da interpretação do Antrl sobre a gramática de reconhecimento do WSMML do apêndice A deste trabalho. Essa capacidade pode ser demonstrada a partir da interpretação gráfica do Antrl IDE sobre os exemplos de WSMML propostos gerando uma árvore sintática conforme mostra a figura abaixo. Às árvores sintáticas completas geradas pela interpretação da ontologia e pela interpretação do *goal* são respectivamente os apêndices C e D deste trabalho.



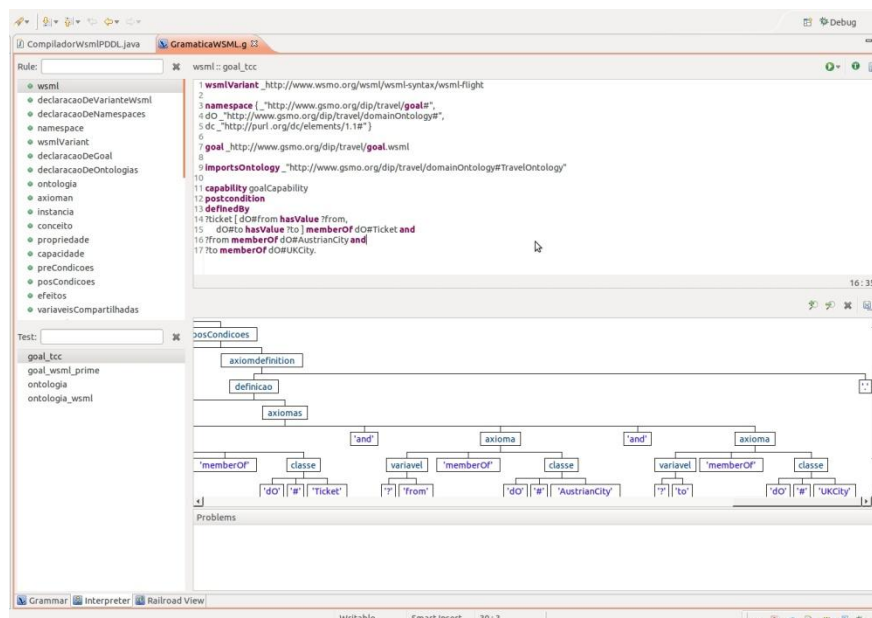


Figura 20 – Árvore sintática produzida pela interpretação do Antlr sobre o *goal*

O processo de tradução do WSM para PDDL provocou a criação da gramática exposta no apêndice B deste trabalho. É dela a responsabilidade de interpretação do WSM e a conseqüente tradução em um artefato equivalente em PDDL. Por sua vez, o resultado gerado através da tradução do *goal* WSM especificado a figura 21 gerou um objetivo em linguagem PDDL conforme ilustra a figura 23. O resultado gerado através da tradução das ontologias WSM especificadas na figura 22 gerou o artefato PDDL ilustrado na figura 24.

```

wsmVariant _http://www.wsmo.org/wsm/wsm-syntax/wsm-flight
namespace { _http://www.gsmo.org/dip/travel/goal#
do _http://www.gsmo.org/dip/travel/domainOntology#
dc _http://purl.org/dc/elements/1.1#
}
goal _http://www.gsmo.org/dip/travel/goal.wsm
importsOntology _http://www.gsmo.org/dip/travel/domainOntology#TravelOntology"
capability goalCapability
postcondition
definedBy
Ticket [ doFrom hasValue ?from
doTo hasValue ?to ] memberOf doTicket and
from memberOf doAustrianCity and
to memberOf doUKCity.

```

Figura 211 – *Goal* WSM especificando uma viagem (FENSEL et al., 2011)

```

wsm:Variant http://www.wsmo.org/wsm/wsm-syntax/wsm-flight

namespace {
  _ "http://www.gsno.org/dip/travel/domainOntology#",
  dc "http://purl.org/dc/elements/1.1#",
  wsm http://www.wsmo.org/wsm/wsm-syntax# }

ontology TravelOntology
  concept Ticket
    annotations
      dc:description hasValue "concept of a ticket"
    endAnnotations
    from ofType Region
    to ofType Region
    vehicle ofType Vehicle

  concept Region

  concept Country subConceptOf Region
    name ofType _string

  concept City subConceptOf Region
    name ofType _string
    country ofType Country

  concept EUCity subConceptOf City

  concept GermanCity subConceptOf EUCity

  concept AustrianCity subConceptOf EUCity

  concept UKCity subConceptOf EUCity

  concept USCity subConceptOf City

  concept Vehicle
    seats ofType _integer

  concept Airplane subConceptOf Vehicle

  concept Train subConceptOf Vehicle

  axiom GermanCityDef
    definedBy
      ?city memberOf GermanCity implies ?city[country hasValue Germany].

  axiom AustrianCityDef
    definedBy
      ?city memberOf AustrianCity impliedBy ?city[name hasValue "Austria"] memberOf country.

  axiom UKCityDef
    definedBy
      ?city memberOf UKCity implies ?city[country hasValue UK].

  instance Innsbruck memberOf AustrianCity

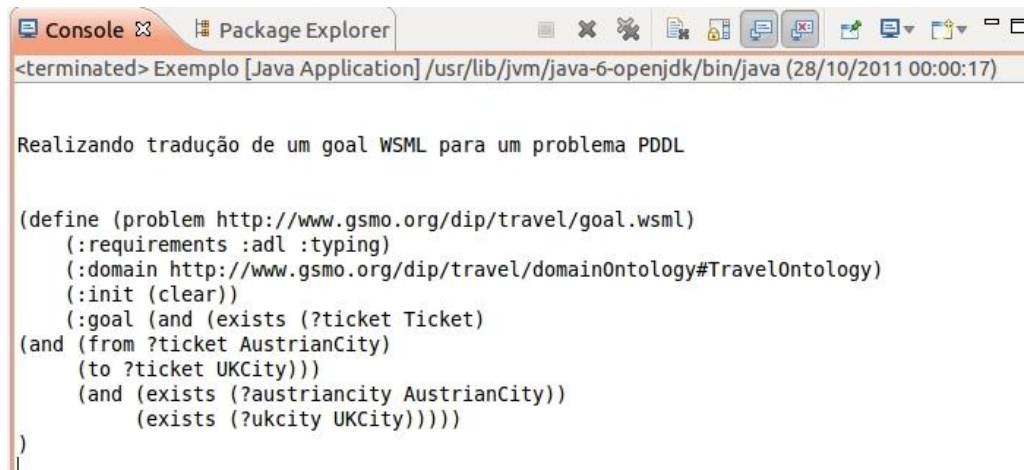
  instance Germany memberOf Country
    name hasValue "Germany"

  instance UK memberOf Country
    name hasValue "United Kingdom"

  instance Austria memberOf Country
    name hasValue ""Austria"

```

Figura 222 – Ontologia de uma viagem em WSM (FENSEL et al., 2011)




```
<terminated> Exemplo [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (28/10/2011 00:00:17)

Realizando tradução de um goal WSML para um problema PDDL

(define (problem http://www.gsml.org/dip/travel/goal.wsml)
  (:requirements :adl :typing)
  (:domain http://www.gsml.org/dip/travel/domainOntology#TravelOntology)
  (:init (clear))
  (:goal (and (exists (?ticket Ticket)
    (and (from ?ticket AustrianCity)
      (to ?ticket UKCity)))
    (and (exists (?austriancity AustrianCity)
      (exists (?ukcity UKCity))))))
  )
)
```

Figura 233 – Tradução de um objetivo WSML para um problema PDDL



```
<terminated> Exemplo [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (28/10/2011 00:21:45)
Realizando tradução de ontologia WSML para um domínio PDDL

(define (domain TravelOntology)
  (:requirements :adl :typing)
  (:types City Country - Region
    UKCity AustrianCity GermanCity - EUCity
    USCity EUCity - City
    Vehicle Region Ticket - Object
    Train Airplane - Vehicle)
  (:predicates (Ticket ?ticket)
    (from ?ticket - Ticket ?region - Region)
    (to ?ticket - Ticket ?region - Region)
    (vehicle ?ticket - Ticket ?vehicle - Vehicle)
    (Region ?region)
    (Country ?country)
    (name ?country - Country ?string - string)
    (City ?city)
    (name ?city - City ?string - string)
    (country ?city - City ?country - Country)
    (EUCity ?eucity)
    (GermanCity ?germancity)
    (AustrianCity ?austriancity)
    (UKCity ?ukcity)
    (USCity ?uscity)
    (Vehicle ?vehicle)
    (seats ?vehicle - Vehicle ?integer - integer)
    (Airplane ?airplane)
    (Train ?train))
)
```

Figura 244 – Tradução de uma ontologia WSML para um domínio PDDL

O resultado da tradução passou então por uma análise que determinou a validade do documento PDDL obtido. Esse processo consistiu no reconhecimento do resultado pela gramática do PDDL no Antlr IDE assim como indicam as figuras 25 e 26.

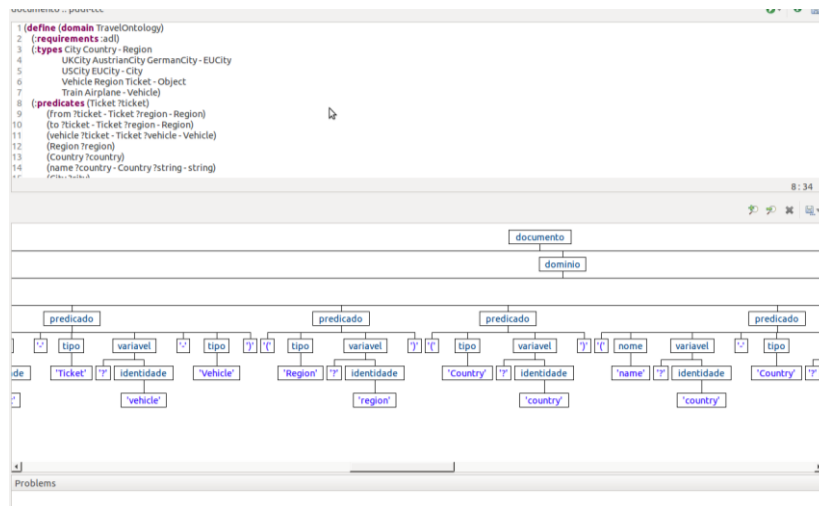


Figura 25 – Reconhecimento do Antlr sobre o PDDL gerado para uma ontologia

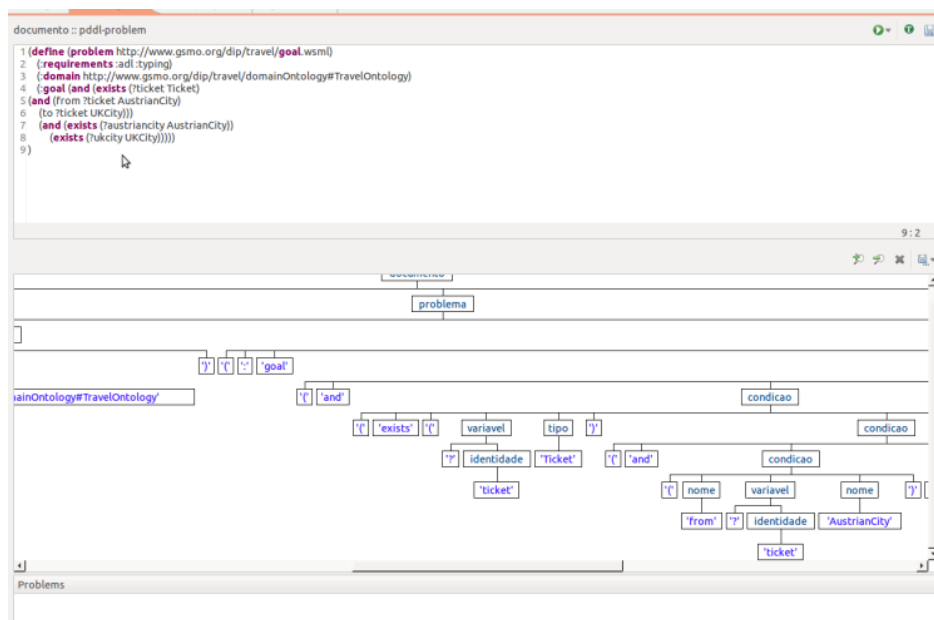


Figura 26 – Reconhecimento do Antlr sobre o PDDL gerado para um goal

O sistema desenvolvido está projetado como um módulo e, portanto, pode ser baixado e acessado através da *internet*. A existência dessa característica permitiu o desenvolvimento de um sistema de tradução *online*<sup>4</sup> que incorpora o projeto desenvolvido neste trabalho e permite a tradução de documentos WSMML a partir da internet.

<sup>4</sup> Acessível em: <http://wsml2pddl.appspot.com>

## 6 Discussão

O processo de desenvolvimento de software frequentemente envolve um balanceamento eficiente entre escopo e tempo para elaboração. Nesse contexto, este trabalho acabou não abrangendo camadas mais complexas tanto da linguagem de entrada WSML como da linguagem-alvo PDDL a tradução de *webservices* e mediadores.

A elaboração de um compilador não é algo trivial e exige grande habilidade e criatividade por parte do seu criador. Pois é demanda conhecimento técnico específico para cada fase da tradução de uma linguagem para outra. Por outro lado, essa dificuldade pode ser reduzida pelo uso dos chamados compiladores de compiladores, como o Antlr e LEX/YACC, que foram desenvolvidos com a intenção de automatizar parte dos ciclos de uma tradução, seja a análise léxica, sintática ou até otimização do código fonte gerado.

O reuso de software estimula a inovação. À medida que uma ferramenta bem sucedida tem sua funcionalidade agregada dentro de um sistema maior passamos então a pesquisar a novidade, evitando de reinventar o existente. Em outras palavras, o uso do Antlr permitiu que muitas fases do compilador fossem facilitadas e o Antlr IDE facilitou a construção de gramáticas e a identificação de árvores sintáticas, pois à medida que as gramáticas eram introduzidas, era gerada uma árvore. A intenção de reuso levou o sistema desenvolvido por este trabalho a ser disponibilizado como um arquivo que pode ser agregado a outros sistemas construídos em Java.

A opção pela pesquisa e implementação de práticas consideradas como exemplos de programação foi mais um ponto de sucesso do projeto. O padrão de projeto *Interpreter* permitiu que o programa crescesse de forma mais organizada e de fácil entendimento ao tratar regras da gramática em classes de programação. Nesse sentido, é de se esperar que outra pessoa que decida por aprimorar o sistema aqui construído tenha uma adaptação mais tranquila. Por outro lado, o uso desse padrão resultou na criação de um elevado número de classes de tratamento de regras e no caso de o PDDL vir a tornar-se uma linguagem mais detalhada, maior será a quantidade de classes geradas para satisfazer o processo de tradução.

Os trabalhos voltados ao desenvolvimento de ferramentas para a *web* semântica ainda estão em processo de criação e amadurecimento e, portanto, ainda carecem de pesquisas, em

especial com o uso da linguagem WSML. Nesse sentido, o trabalho aqui desenvolvido é inovador porque incentiva o uso da linguagem WSML para a próxima geração de serviços ao contrário de outros tradutores que incentivam a tradução de OWL, RDF Schema ou DAML para PDDL. Por outro lado, a linguagem WSML, ao contrário dos outros modelos, talvez por ser a mais recente, tem poucos exemplos concretos e completos, o que é um obstáculo a sua difusão na área.

Por fim, a colaboração é extremamente importante em trabalhos principalmente em áreas de pesquisas pioneiras. Essa preocupação, aliada ao objetivo de tornar-se um trabalho de referência, incentivou com que todo o desenvolvimento do sistema fosse sistematicamente controlado em um repositório de código fontes aberto e colaborativo chamado GitHub onde as mudanças poderão ser consultadas através da *internet* e o sistema continuado a baixo custo. Ainda visando facilitar o processo de desenvolvimento de trabalhos futuros, foi disponibilizado um sistema *web* que utiliza em seu coração o sistema produzido neste trabalho para realizar traduções de ontologias e objetivos pela *web*.

## 7 Conclusão e diretrizes para trabalhos futuros

Este trabalho apresenta um sistema que, apesar de ainda apresentar algumas limitações, é capaz de conectar o conceito de *web service* semântico com o de um problema planejamento. Nesse sentido, este trabalho é uma contribuição a pesquisa sobre *web services* semânticos e também excita trabalhos sobre a composição automática de serviços com agregação de significado a partir da aplicação de técnicas de inteligência artificial.

A construção de um compilador demonstrou que é possível realizar a tradução de uma linguagem *web* semântica para uma linguagem de planejamento por uma abordagem diferente da utilizada por outros trabalhos que utilizam tabelas de equivalências. Nesse contexto, este trabalho ainda iluminou a possibilidade de traduzir não somente OWL para PDDL (DOU et al., 2005) ou uma variação de linguagem *web* semântica como o tradutor DAML para PDDL (PDDAML, 2011), mas também WMSL para PDDL e isso incentiva uma nova área de pesquisas na *web* semântica, pois WMSL é uma linguagem mais rica que OWL.

A opção pelo uso de ferramentas já existentes foi um fator decisivo no sucesso desta monografia. A escolha pelo Eclipse e pela extensão Antlr IDE facilitou muito a interação com a linguagem Java com a ferramenta do ANTLR desenvolvida por Terence Parr. Já a ferramenta de Parr tornou mais simples a criação de analisadores léxicos, sintáticos, semânticos e a geração de código fonte. Em outras palavras, o reuso de sistemas auxilia o foco no novo, naquilo que ainda não existe, e leva a obtenção de resultados finais mais expressivos.

Ao passo em que o reaproveitamento de código fonte é importante, a colaboração entre indivíduos trabalhando para um mesmo fim também o é. Espera-se que com a adoção de linguagem de programação acessível, orientação a objetos, *design patterns* e a viabilização de um controlador de versão aberto estimulem simpatizantes a participarem do aperfeiçoamento do sistema proposto nesse trabalho.

Por fim, o objetivo principal deste trabalho foi atingido e o mecanismo tradutor de metas WMSL em PDDL foi construído. Deseja-se que novas pesquisas possam ser realizadas utilizando esta monografia como parte da referência. Algumas sugestões de trabalhos futuros

que poderiam dar continuidade a este são: o estudo de técnicas mais eficientes da tradução de domínios, maneiras de tradução de WSML para PDDL usando mediadores, a tradução de um *webservice* WSML para PDDL entre outros trabalhos que envolvem tradução de linguagens, *web* semântica e tarefas de planejamento.



## 8 Referências Bibliográficas

ANTLR IDE. Disponível em: <<http://antlr3ide.sourceforge.net/>>. Acesso em: 18 Out. 2011.

AHO, A.; LAM, M.; SETHI, R.; ULLMAN, J. **Compilers: Principles, Techniques, & Tools**, Ed. Addison Wesley, 2007, 2nd edition, 1008 p.

ALESSO, H.; SMITH C. **Developing Semantic Web Services**, Ed. A K Peters Ltd., 2005, 445 p.

ALONSO, G.; CASATI, F.; KUNO, H.; MACHIRAJU, V. **Web Services: Concepts, Architectures and Applications**, Ed. Springer, 2004, 354 p.

BENACCHIO, J. H. Q. **Planejamento em inteligência artificial utilizando rede de perti cíclicas**. Curitiba: Universidade Federal do Paraná, 2008. 97 p.

BRUIJN, J.; FENSEL, D.; KERRIGAN, M.; KELLER, U.; LAUSEN, H.; SCICLUNA, J. **Modeling Semantic Web Services: The Web Service Modeling Language**, Ed. Springer, 2008, 192 p.

COOPER, K.; TORCZON, L. **Engineering a compiler**, Ed. Morgan & Kaufmann, 2004, 801 p.

DIGIAMPIETRI, L. A. **Gerenciamento de workflows científicos em bioinformática**. 2007. 88 f. Tese (Doutorado em ciência da computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2007.

DIGIAMPIETRI, L. A.; PÉREZ-ALCÁZAR, J.J.; MEDEIROS, C. B. AI Planning in Web Services Composition: a review of current approaches and a new solution. In: VI Encontro Nacional de Inteligência Artificial – ENIA, 2007, Rio de Janeiro. **ANAIS DO XXVII CONGRESSO DA SBC**, Rio de Janeiro, 2007.

DOU, D.; MCDERMOTT, D.; QI, PEISHEN. **Ontology Translation on the Semantic Web**. New York, 2005. Disponível em: <[http://cs-www.cs.yale.edu/homes/ddj/papers/ontomerge\\_jds.pdf](http://cs-www.cs.yale.edu/homes/ddj/papers/ontomerge_jds.pdf)>. Acesso em: 23 Dez, 2011.

FENSEL, D.; KERRIGAN, M.; ZAREMBA, M. **Implementing Semantic Web Services: The SESA Framework**, Ed. Springer, 2007, 322 p.

FENSEL, D.; LAUSEN, H.; POLLERES, A.; BRUIJN, J.; STOLLBERG, M.; ROMAN, D., DOMINGUE, J. **Enabling Semantic Web Services: The Web Service Modeling Language**, Ed. Springer, 2007, 188 p.

FENSEL, D.; FACCA, F.; SIMPERL, E.; TOMA, I. **Semantic Web Services**, Ed. Springer, 2011, 368p.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos**, Ed. Artmed, 2005, 364 p.

GHALLAB, M.; HOWE, A.; KNOBLOCK, C.; MCDERMOTT, D.; RAM, A.; VELOSO, M.; WELD, D.; WILKINS, D. PDDL the planning domain definition language. In Proceeding of AIPS-98 Planning Committee, 1998.

GHALLAB, M.; NAU, D.; TRAVERSO, P.; **Automated Planning: Theory and practice**, Ed. Morgan Kaufmann, 2nd Edition 2004, 635 p.

HEBELER, J.; FISHER M.; BLACE, R.; PEREZ-LOPEZ, A. **Semantic Web Programming**, Ed. Wiley; 1st Edition, 2009, 648 p.

KASHYAP V.; BUSSLER, C.; MORAN M. **The Semantic Web Services: Semantics for Data and Services on the Web**, Ed. Springer, 2008, 414 p.

LOUDEN, K. **Compiladores: Princípios e práticas**, Ed. Thomson, 2004, 569 p.

PARR, T.; **The Definitive ANTLR Reference: Building Domain-Specific Languages**, 2007, 376 p.

PÉREZ-ALCÁZAR, J. J. **Projeto: O uso de Planejamento em Inteligência Artificial para a Composição Automática de Serviços Web**, São Paulo: EACH-USP, 2007. 25 p. Relatório científico apresentado à FAPESP, São Paulo.

PDDAML. An Automatic Translator between PDDL and DAML. Disponível em: <[http://www.cs.yale.edu/homes/dvm/daml/pddl\\_daml\\_translator1.html](http://www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator1.html)>. Acesso em: 23 dez. 2011.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**, Ed. Campus, 2004, 1021 p.

SAFONOV, V. **Trustworthy Compilers**, Ed. John Wiley and Sons, 2010, 295 p.

## 9 APÊNDICE A – Gramática da linguagem WSMML

```
grammar GramaticaWSMML;

options {
    language = Java;
    output=AST;
    ASTLabelType=CommonTree;
}

tokens {
    WsmmlVariant;
}

@header {
    package br.usp.each.wsmml2pddl.gramaticas;
}

@lexer::header {
    package br.usp.each.wsmml2pddl.gramaticas;
}

wsmml
    : declaracaoDeVarianteWsmml?
      declaracaoDeNamespaces?
      ( declaracaoDeOntologias | declaracaoDeGoal )
    ;

declaracaoDeVarianteWsmml
    : wsmmlVariant fullIri
    ;

declaracaoDeNamespaces
    : 'namespace' ( namespace | '{' namespace (',' namespace)* '}' )
    ;

namespace
    : Identificador? fullIri
    ;

wsmmlVariant
    : 'wsmmlVariant' -> WsmmlVariant
    ;

declaracaoDeGoal
    : 'goal' fullIri?
      anotacoes?
      ontologiaImportada?
      capacidade
    ;

declaracaoDeOntologias
    : ontologia*
    ;

ontologia
    : 'ontology' Identificador
      conceito+
    ;

conceito
    : 'concept' Identificador ('subConceptOf' Identificador)?
```

```

                                anotacoes?
                                (Identificador 'ofType' Identificador )*
                                ;

capacidade
    : 'capability' fullIri?
      variaveisCompartilhadas?
      ( preCondicoes | posCondicoes | efeitos )*
    ;

preCondicoes
    :      'precondition'
      axiomdefinition
    ;

posCondicoes
    :      'postcondition'
      axiomdefinition
    ;

efeitos
    :      'effect'
      axiomdefinition
    ;

variaveisCompartilhadas
    : 'sharedVariables' '{' variavel (',' variavel)* '}'
    ;

axiomdefinition
    :      anotacoes?
      definicao '.'
    ;

definicao
    : 'definedBy'
      axiomas*
    ;

axiomas
    : axioma ('and' axioma)*
    ;

axioma
    : variavel ('[' propriedades ']')? 'memberOf' classe
    ;

propriedades
    : atributo (',' atributo)*
    ;

classe
    : (Identificador '#')? Identificador
    ;

variavel
    : '?' Identificador
    ;

anotacoes
    :      'annotations'
      atributo*
      'endAnnotations'
    ;

```

```

ontologiaImportada
    : 'importsOntology'
        ( fullIri
        | '{' fullIri (',' fullIri)*      '}' )
    ;

atributo
    : classe 'hasValue' (variavel | StringLiteral )
    ;

fullIri
    : ' ' StringLiteral
    | Identificador
    ;

prefixo
    : Identificador fullIri
    ;

fragment AlfaNumerico          : Digito | Letra;
fragment Digito                 : '0'..'9';
fragment Espacamento          : ' ';
fragment Letra                  : 'a'..'z' | 'A'..'Z';
fragment QuebraDeLinha         : '\n' | '\r';
fragment Tabulacao             : '\t';

Inteiro : Digito+;

Identificador: (Letra)(AlfaNumerico | '-' )*;

StringLiteral
    : '"'
        { StringBuilder b = new StringBuilder(); }
        ( '\\ ' '"'
        { b.appendCodePoint('"'); }
        | c = ~( '" ' ) { b.appendCodePoint(c); }
        )*
        '"'
        { setText(b.toString()); }
    ;

CaracteresIgnorados: (Espacamento | Tabulacao | QuebraDeLinha | '\f')+ {$channel =
HIDDEN;};
ComentarioDeLinhaUnica: '//' .* (QuebraDeLinha) {$channel = HIDDEN;};
ComentarioDeMultiplasLinhas: '/*' .* '*/' {$channel = HIDDEN;};

```

## 10 APÊNDICE B – Gramática do tradutor PDDL

```
tree grammar TradutorWSML2PDDL;

options {
    language = Java;
    tokenVocab = GramaticaWSML;
    ASTLabelType = CommonTree;
}

@header {
    package br.usp.each.wsml2pddl.gramaticas;
    import br.usp.each.wsml2pddl.modelo.avaliadores.Avaliador;
    import java.util.HashMap;
    import java.util.Map;
    import java.util.List;
    import java.util.ArrayList;
    import br.usp.each.wsml2pddl.avaliadores.*;
    import org.antlr.stringtemplate.StringTemplate;
}

@members{

    Map<String, String> propriedades = new HashMap<String, String>();
    Map<String, List<String>> superclasses = new HashMap<String,
List<String>>();
    List<Avaliador> predicados = new ArrayList<Avaliador>();

}

avaliador returns [Avaliador e]
    :      documento EOF { $e = new AvaliadorDeDocumentoPDDL($documento.e); }
    ;

documento returns [Avaliador e]
    : varianteWsml?
      prefixosImportados?
      (declaracaoDeOntologias { $e = $declaracaoDeOntologias.e; })?
      (declaracaoDoGoal { $e = $declaracaoDoGoal.e; })?
    ;

declaracaoDeOntologias returns [Avaliador e]
    : 'ontology' string
      conceito*
      {
          final Avaliador dominio = $string.e;
          final Avaliador requerimentos = new
AvaliadorDeRequerimentos();
          final Avaliador tipos = new
AvaliadorDeTipos(superclasses);
          final Avaliador predicado = new
AvaliadorDePredicados(predicados);
          $e = new AvaliadorDeDominio(dominio, requerimentos, tipos,
predicado);
      }
    ;
```

```

conceito
    : 'concept' c1 = Identificador
      ('subConceptOf' c2 = Identificador )?
      {
          predicados.add(new AvaliadorDePredicado(new
AvaliadorDeString($c1.text)));
          final String superClasse = ($c2 != null)? $c2.text
: "Object";
          final List<String> objetosDaClasseC2 =
superclasses.get(superClasse);
          if (objetosDaClasseC2 == null){
              superclasses.put(superClasse, new
ArrayList<String>());
              superclasses.get(superClasse).add($c1.text);
          } else {
              objetosDaClasseC2.add($c1.text);
          }
          anotacoes?
          ( p = Identificador 'ofType' c = Identificador
            {
                predicados.add(new
AvaliadorDePredicado(new AvaliadorDeString($c1.text),
                        new AvaliadorDeString($p.text),
                        new AvaliadorDeString($c.text))); }
          ) *
      ;

propriedade returns [Avaliador e]
    : classe 'hasValue' variavel
      {
          e = new AvaliadorDePropriedade($classe.e);
      }
    ;

anotacoes
    :
      'annotations'
      propriedade+
      'endAnnotations'
    ;

declaracaoDoGoal returns [Avaliador e]
    :
      'goal' fullIri
      importsOntology?
      condicoesDoProblema
      {
          Avaliador avaliadorDeProblema = new
AvaliadorDeString($fullIri.e.avalia());
          Avaliador avaliadorDeDominio = new
AvaliadorDeDeclaracaoDeDominio($fullIri.e);
          Avaliador avaliadorDeObjetos = new
AvaliadorDeObjetos(null);
          Avaliador avaliadorDeEstadoInicial = new
AvaliadorDeEstadoInicial(null);
          Avaliador avaliadorDeRequerimentos = new
AvaliadorDeRequerimentos();
          $e = new AvaliadorDeProblema(avaliadorDeProblema,
avaliadorDeRequerimentos, avaliadorDeDominio,
avaliadorDeObjetos,
avaliadorDeEstadoInicial, $condicoesDoProblema.e);
      }
    ;

prefixosImportados

```



```

        : 'namespace' ( namespace | '{' namespace (',' namespace)* '}' )
        ;

varianteWsmml
    : WsmmlVariant fullIri
    ;

importsOntology
    : 'importsOntology'
      ( fullIri | '{' fullIri (',' fullIri)* '}' )
    ;

condicoesDoProblema returns [Avaliador e]
    : 'capability'
      (posCondicoes = postconditions)
      { $e = new AvaliadorDeGoal($posCondicoes.e); }
    ;

postconditions returns [Avaliador e]
    : 'postcondition'
      'definedBy'
      axioma
      {
          $e = new
AvaliadorDePosCondicoes($axioma.e);
          for(final String propriedade:
propriedades.keySet()){
              $e = new AvaliadorDeCondicao($e,
propriedade, propriedades.get(propriedade));
          }
      }
      '.'
    ;

axioma returns [Avaliador e]
    : c1 = condicao { $e = $c1.e; }
      ( 'and' c2 = axioma { $e = new AvaliadorAnd($c1.e, $c2.e); } )?
    ;

condicao returns [Avaliador e]
    : variavel ('[' propriedades ']')? 'memberOf' classe
      {
          propriedades.put($variavel.e.avaliao(),
$classe.e.avaliao());
          e = new AvaliadorExists(new AvaliadorDeClasse($classe.e,
$propriedades.e));
      }
    ;

propriedades returns [Avaliador e]
    : p1 = propriedade { e = $p1.e; }
      (',' p2 = propriedade { e = new AvaliadorAnd($p1.e, $p2.e); } )*
    ;

classe returns [Avaliador e]
    : v1 = Identificador '#' v2 = Identificador { e = new
AvaliadorDeString($v2.text); }
    ;

variavel returns [Avaliador e]
    : '?'? string { e = new AvaliadorDeVariavel($string.e); }
    ;

atributo returns [Avaliador e]

```

```

        : Identificador { e = new AvaliadorDeString($Identificador.text); }
    ;

fullIri returns [Avaliador e]
    : '_'? valor { $e = $valor.e; }
    ;

valor returns [Avaliador e]
    : string { e = $string.e; }
    ;

string returns [Avaliador e]
    : StringLiteral { e = new AvaliadorDeString($StringLiteral.text); }
    | Identificador { e = new AvaliadorDeString($Identificador.text); }
    ;

namespace
    : Identificador? fullIri
    ;

```

## **11 APÊNDICE C– Árvore sintática de uma ontologia**



## 12 APÊNDICE D – Árvore sintática de um *goal*



## 13 APÊNDICE E – Gramática de verificação do PDDL

```
grammar PDDL;

options {
    language = Java;
    output=AST;
    ASTLabelType=CommonTree;
    //backtrack=true;
}

tokens {
    DOMAIN;
}

@header {
    package br.usp.each.wsml2pddl.gramaticas;
}

@lexer::header {
    package br.usp.each.wsml2pddl.gramaticas;
}

documento
    : '(' 'define'
      ( dominio | problema )
      ','
;

dominio
    : declaracaoDeDominio
      requerimentos?
      tipos?
      predicados?
      acao*
;

declaracaoDeDominio
    : '(' ':'? 'domain' (nome | url) ')'
;

declaracaoDeProblema
    : '(' 'problem' (nome | url) ')'
;

problema
    : declaracaoDeProblema
      requerimentos?
      declaracaoDeDominio
      objetos?
      estadoInicial?
      objetivo?
;

requerimentos
    : '(' ':' 'requirements' requerimento+ ')'
;

objetos
    : '(' ':' 'objects' nome+ ')'
;

tipos
    : '(' ':' 'types'
      tipo+
      ')'
;
```

```

tipo
    : Identidade '-'?
    ;

estadoInicial
    : '(' ':' 'init' estado+ ')'
    ;

objetivo
    : '(' ':' 'goal' condicao ')'
    ;

requerimento
    : ':adl'
    | ':strips'
    | ':typing'
    | ':conditional-effects'
    | ':existential-preconditions'
    ;

predicados
    : '(' ':' 'predicates' predicado+ ')'
    ;

predicado
    : '(' tipo variavel+ ')'
    | '(' nome variavel '-' tipo variavel '-' tipo ')'
    ;

acao
    : '(' ':' 'action' nome
        parametros
        precondicoes?
        efeitos
        ')'
    ;

parametros
    : ':' 'parameters' '(' variavel+ ('-' tipo)? ')'
    ;

precondicoes
    : ':' 'precondition' condicao
    ;

efeitos
    : ':' 'effect' condicao
    ;

estado
    : '(' nome nome nome? ')'
    ;

condicao
    : '(' 'and' condicao+ ')'
    | '(' 'not' condicao+ ')'
    | '(' 'forall' '(' variavel '-' tipo ')' condicao+ ')'
    | '(' 'when' condicao condicao ')'
    | '(' 'exists' '(' variavel tipo ')' condicao* ')'
    | '(' nome (variavel | nome)+ ')'
    ;

variavel
    : '?' identidade
    ;

identidade
    : Identidade
    ;

nome

```

```

        : Identidade
        ;

url
    : Url
    ;

fragment AlfaNumerico          : Digito | Letra;
fragment Digito                 : '0'..'9';
fragment Espacamento          : ' ';
fragment Letra                  : 'a'..'z' | 'A'..'Z';
fragment QuebraDeLinha         : '\n' | '\r' | '\r\n' | '\n\r';
fragment Tabulacao              : '\t';

Inteiro : Digito+;

Identidade: (Letra) (AlfaNumerico | '-' | '_' )*;

Url: ('http://') (AlfaNumerico) (AlfaNumerico | '-' | '/' | ';' | '#' | '.' )*;

StringLiteral
    : '"'
      {
        { StringBuilder b = new StringBuilder(); }
        { '\\ ' '"' { b.appendCodePoint(' '); }
        | c = ~( ' ' ) { b.appendCodePoint(c); }
        } *
        '"'
        { setText(b.toString()); }
      ;

CaracteresIgnorados: (Espacamento | Tabulacao | QuebraDeLinha | '\f') + { $channel = HIDDEN; };
ComentarioDeLinhaUnica: ';' .* (QuebraDeLinha) { $channel = HIDDEN; };

```



## 14 ANEXO A – Exemplo de ontologia em WSMML

**wsmmlVariant** <http://www.wsmo.org/wsmml/wsmml-syntax/wsmml-flight>

**namespace** { \_ "http://www.gsml.org/dip/travel/domainOntology#",  
dc \_ "http://purl.org/dc/elements/1.1#",  
wsmml \_ <http://www.wsmo.org/wsmml/wsmml-syntax#> }

**ontology** TravelOntology

**concept** Ticket  
  annotations  
    dc#description hasValue "concept of a ticket "  
  endAnnotations  
  from ofType Region  
  to ofType Region  
  vehicle ofType Vehicle

**concept** Region

**concept** Country **subConceptOf** Region  
  name ofType \_string

**concept** City **subConceptOf** Region  
  name ofType \_string  
  country ofType Country

**concept** EUCity **subConceptOf** City

**concept** GermanCity **subConceptOf** EUCity

**concept** AustrianCity **subConceptOf** EUCity

**concept** UKCity **subConceptOf** EUCity

**concept** USCity **subConceptOf** City

**concept** Vehicle  
  seats ofType \_integer

**concept** Airplane **subConceptOf** Vehicle

**concept** Train **subConceptOf** Vehicle

**axiom** GermanCityDef  
  **definedBy**  
    ?city memberOf GermanCity implies ?city[country hasValue Germany].

**axiom** AustrianCityDef  
  **definedBy**  
    ?city memberOf AustrianCity impliedBy ?city[name hasValue "Austria"] memberOf country.

**axiom** UKCityDef  
  **definedBy**  
    ?city memberOf UKCity implies ?city[country hasValue UK].

**instance** Innsbruck memberOf AustrianCity

**instance** Germany memberOf Country  
  name hasValue "Germany"

**instance** UK memberOf Country  
  name hasValue "United Kingdom"

**instance** Austria memberOf Country  
  name hasValue "Austria"

## 15 ANEXO B – Exemplo de *goal* descrito em WMSL

**wsmIVariant** <http://www.wsmo.org/wsmI/wsmI-syntax/wsmI-flight>

**namespace** { \_ "http://www.gsmo.org/dip/travel/goal#",  
dO \_ "http://www.gsmo.org/dip/travel/domainOntology#",  
dc <http://purl.org/dc/elements/1.1#> }

**goal** \_ "http://www.gsmo.org/dip/travel/goal.wsmI"

**importsOntology** \_ "http://www.gsmo.org/dip/travel/domainOntology#TravelOntology"

**capability** goalCapability

**postcondition**

**definedBy**

?ticket [ dO#from hasValue ?from,  
dO#to hasValue ?to ] memberOf dO#Ticket and  
?from memberOf dO#AustrianCity and  
?to memberOf dO#UKCity.