UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL DE CONHECIMENTO

WALKER ARAÚJO ATAIDE

UM MODELO SEMÂNTICO PARA ENGENHARIA DE APLICAÇÃO DE LINHAS DE PRODUTO DE SOFTWARE PARA SISTEMAS TUTORES INTELIGENTES

MACEIÓ
2015

Walker Araújo Ataide

Um modelo semântico para engenharia de aplicação de linhas de produto de software para sistemas tutores inteligentes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas como requisito parcial para obtenção do grau de Mestre em Modelagem Computacional de Conhecimento .

Orientador: Prof. Dr. Alan Pedro da Silva
Orientador: Prof. Dr. Patrick Henrique da Silva Brito

Maceió
2015

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecário Responsável: Valter dos Santos Andrade

A862p Ataide, Walker Araújo.
Um modelo semântico para engenharia de aplicação de linhas de produto de software para sistemas tutores inteligentes / Walker Araújo Ataide. – 2015.
145 f. : il.

Orientador: Alan Pedro da Silva.
Orientador: Patrick Henrique da Silva Brito.
Dissertação (Mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Programa de Pós-Graduação em Modelagem Computacional de Conhecimento. Maceió, 2015.

Bibliografia: f. 102-109.
Apêndices: f. 110-145.

1. Sistemas tutores inteligentes. 2. Linha de produto - Software. 3. Software de aplicação - Desenvolvimento. 4. Ontologia. I. Título.

CDU: 004.78

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 09, Cidade Universitária
CEP 57.072-900 – Maceió – AL – Brasil
Telefone: (082) 3214-1364

MODELAGEM COMPUTACIONAL DE CONHECIMENTO

Membros da Comissão Julgadora da Dissertação de Mestrado de Walker Araújo Ataíde, intitulada: "Um modelo semântico para engenharia de aplicação de linhas de produto de software para sistemas tutores inteligentes.", apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas, em quatro de agosto de 2015, às 10h00min, na sala 33 do Instituto de Computação.

COMISSÃO JULGADORA

Prof. Dr. Alan Pedro da Silva
UFAL – Instituto de Computação
Orientador

Prof. Dr. Patrick Henrique da Silva Brito
UFAL – Instituto de Computação
Orientador

Prof. Dr. Ig Ibert Bittencourt Santana Pinto
UFAL – Instituto de Computação
Examinador

Prof. Dr. Ivo Augusto Andrade Rocha Calado
IFAL – Unidade Palmeira dos Índios
Examinador

Prof. Dr. Sean Wolfgand Matsui Siqueira
UNIRIO – Centro de Ciências Exatas e Tecnologia
Examinador

Maceió, agosto de 2015.

À minha mãe que me deu todo suporte moral e material para que eu pudesse concretizar meus sonhos e a minha esposa e companheira que me fortalece com seu amor e apoio em todos os momentos.

AGRADECIMENTOS

Sou muito grato aos meus orientadores, Prof. Dr. Patrick Henrique da Silva Brito e Prof. Dr. Alan Pedro da Silva, pedras fundamentais desta dissertação e colaboradores diretos de minha formação acadêmica, pela atenção, paciência, confiança e amizade.

Agradeço a equipe do grupo de pesquisa NEES - Núcleo de Excelência em Tecnologias Sociais que colaborou com este trabalho, especialmente: Olavo de Holanda Cavalcanti Neto, Thyago Tenório Martins de Oliveira, Jean Melo e Prof. Dr. Ig Ibert Bittencourt.

Tudo tem o seu tempo determinado, e há tempo para todo o propósito debaixo do céu.

Há tempo de nascer, e tempo de morrer; tempo de plantar, e tempo de arrancar o que se plantou;

Tempo de matar, e tempo de curar; tempo de derrubar, e tempo de edificar;

Tempo de chorar, e tempo de rir; tempo de prantear, e tempo de dançar;

Tempo de espalhar pedras, e tempo de ajuntar pedras; tempo de abraçar, e tempo de afastar-se de abraçar;

Tempo de buscar, e tempo de perder; tempo de guardar, e tempo de lançar fora;

Tempo de rasgar, e tempo de coser; tempo de estar calado, e tempo de falar;

Tempo de amar, e tempo de odiar; tempo de guerra, e tempo de paz.

Salomão, Bíblia Sagrada. Eclesiastes 3:1-8.

RESUMO

Sistemas Tutores Inteligentes (STIs) são softwares que buscam representar o comportamento humano inerente ao processo de ensino em algum domínio específico com o objetivo de dar suporte às atividades de professores e oferecer um ensino adaptado aos estudantes. Os STIs têm grande potencialidade tanto no ensino presencial quanto a distância, entretanto, sua construção é uma tarefa complexa e dispendiosa que demanda a presença de profissionais especializados em computação e domínio do sistema sendo agravado quando se necessita construir STIs em larga escala e adaptados a cada domínio. Nesse sentido, a abordagem de Linhas de Produto de Software (LPS) possibilita construir STIs em larga escala. De forma complementar, ontologias podem ser utilizadas para permitir que tal construção seja automaticamente adaptável para diferentes domínios. Porém, na fase de engenharia da aplicação da LPS, momento em que os STIs são instanciados, faz-se necessária a manipulação de ontologias e artefatos de software distintos, demandando a presença de profissionais com conhecimentos em ontologia e engenharia de software, o que dificulta a realização dessa tarefa por autores/projetistas. Com base no exposto propõe-se neste trabalho um modelo baseado em ontologia para automatizar a engenharia de aplicação de LPS para sistemas tutores inteligentes. De maneira específica pretende-se automatizar o processo de customização, instanciação e implantação de STIs de uma LPS, tornando transparente ao usuário a manipulação de ontologias e artefatos de software. O modelo proposto utiliza ontologias para representar as funcionalidades e restrições de uma LPS genérica, as funcionalidades específicas para STIs, as decisões do autor em termos de quais funcionalidades farão parte do STI a ser gerado e as informações do aluno. O processo de engenharia de aplicação da LPS é realizado por componentes que conduzem o autor pelas etapas de autenticação no sistema, seleção da LPS a ser instanciada, customização/configuração das funcionalidades do STI, validação, geração e implantação de um STI em um servidor Web. Para validar o modelo proposto foi construída uma ferramenta que automatiza a geração de produtos em uma LPS. Tal ferramenta foi utilizada em um estudo de caso abrangendo a engenharia de aplicação de um STI a partir de uma LPS. Os resultados obtidos mostram-se adequados apontando como principais contribuições a concepção e desenvolvimento de um modelo semântico para a engenharia de aplicação de LPS para STIs, este modelo guia o autor pelo processo tornando transparente o uso de ontologias e LPS, auxilia na redução da complexidade e do esforço empregado (i.e., carga de trabalho) na construção de STIs a partir de LPS semântica, reduz as qualificações exigidas para instanciar STIs ao qual pode possibilitar que mais pessoas realizem essa tarefa e permite validar corretamente a configuração das funcionalidades do STI a ser instanciado de forma que apenas produtos sem erros de configuração sejam gerados.

Palavras-chave: Sistemas Tutores Inteligentes. Linha de Produto de Software. Engenharia de Aplicação. Ontologia.

ABSTRACT

Intelligent Tutoring Systems (ITSs) are softwares that aims to represent human behavior inherent in the teaching process in any particular field in order to support the activities of teachers and offer a adapted teaching to students. ITSs have great potential both in the classroom teaching as the distance, however, its construction is a complex and expensive task that requires the presence of specialized professionals in computing and system domain being compounded when you need to build ITSs large-scale and adapted to each area. In this sense, the approach Software Product Lines (SPL) allows to build large-scale ITS. Complementarily, ontologies can be used to allow such a construction is automatically adaptable to different domains. However, in the application engineering phase of the SPL, when ITS are instantiated, is required manipulation of ontologies and different software artifacts, demanding the presence of professionals with expertise in ontology and software engineering, making it difficult the accomplishment of this task by authors / designers. Given the above, we propose an ontology-based model to automate the SPL application engineering for intelligent tutoring systems. Specifically, it is intended to automate the processes of customization, instantiation and deployment of an ITS of a SPL, making the manipulation of ontologies and software artifacts transparent to the user. The proposed model uses ontologies to represent the features and constraints of a generic SPL, the specific features for ITSs, the decisions of the author in terms of which features will be part of the ITS to be generated and the information of the student. The SPL application engineering process is performed by components that lead author by the steps of authentication in the system, selection of the SPL to be instantiated, customization/configuration of the features of ITS, validation, generation and deployment of ITS on a Web server. In order to validate the proposed model has been built a tool that automates the generation of products in a SPL. This tool was used on a case study involving the application engineering of an ITS from a SPL. The obtained results showed to be adequate singled out as major contributions, the design and development of a semantic model for the SPL application engineering for ITSs, this model guides the author through the process making transparent the use of ontologies and SPL, helps reduce complexity and effort (i.e., workload) in the construction of ITSs from semantic SPL, reduces the skills required to instantiate ITSs what can enable more people to perform this task and allows properly validate the configuration of the features of the ITS to be instantiated, allowing only products without misconfigurations can be generated.

Keywords: Intelligent Tutoring Systems. Software Product Lines. Application Engineering. Ontology.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BD	Banco de Dados
CAI	*Computer Assisted Instruction*
CRUD	*Create, Read, Update e Delete*
CTAT	*Cognitive Tutor Authoring Tools*
DAO	*Data Access Object*
DB	*Data Base*
EA	Engenharia de Aplicação
EAD	Educação a Distância
ED	Engenharia de Domínio
ELPS	Engenharia de Linha de Produto de Software
FODA	*Feature Oriented Domain Analysis*
GROW	Grupo de Otimização da Web
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
IA	Inteligência Artificial
IAC	Instruções Assistidas por Computador
IEEE	*Institute of Electrical and Electronics Engineers*
ITEE	*IEEE Technology and Engineering Education*
ITS	*Intelligent Tutoring System*
JAR	*Java Archive*
JESS	*Java Expert System Shell*
JOINT	*Java Ontology INtegrated Toolkit*
JSF	*JavaServer Faces*
JSP	*JavaServer Pages*
KAO	*Knowledge Access Object*
LPS	Linha de Produto de Software
LPSS	Linha de Produto de Software Semântica
MVC	*Model View Controller*
NEES	Núcleo de Excelência em Tecnologias Sociais
OWL	*Ontology Web Language*

PDM	*Product Decision Model*
TIC	Tecnologia da Educação e Comunicação
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
RF	Requisito Funcional
RNF	Requisito Não Funcional
S3T	*International Conference on Software, Services and Semantic Technologies*
SGBD	Sistema Gerenciador de Banco de Dados
SPL	*Software Product Line*
STI	Sistema Tutor Inteligente
SWRL	*Semantic Web Rule Language*
UFAL	Universidade Federal de Alagoas
URI	*Uniform Resource Identifier*
URL	*Universal Resource Location*
W3C	*World Wide Web Consortium*
WAR	*Web Application Archive*
WEB	*World Wide Web*
WSWEd	*Workshop on Semantic Web and Education*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a contextualização e a motivação para o investimento em pesquisa a respeito da construção de sistemas tutores inteligentes (STIs), enfatizando a importância de utilizar abordagens que permitam produzir estes sistemas em larga escala, de forma customizada, com maior reuso de artefatos de software e de conhecimento. Além disso, este capítulo apresenta os objetivos e a estrutura desta dissertação.

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO

As Tecnologias da Informação e Comunicação (TICs) vem influenciando profundamente os vários setores da sociedade (e.g., comércio, saúde, ciência, política, educação) e se integrando cada vez mais ao cotidiano da maioria das pessoas (LUZ et al., 2015). Na educação, as TICs têm um potencial elevado de impulsionar o desenvolvimento dos estudantes através da ampliação do acesso à educação, uma vez que a aprendizagem pode acontecer em qualquer lugar e a qualquer momento, e do apoio ao processo de ensino e aprendizagem, desde que aliadas a boas estratégias pedagógicas (COSTA, 2011; FU, 2013). Como exemplo, o uso de plataformas de gestão de aprendizagem podem possibilitar a realização de um conjunto diversificado de atividades em contextos de aprendizagem favorecendo a construção de novos cenários educacionais (MORAIS et al., 2014).

Nos sistemas de software educacionais tradicionais o computador geralmente responde a cada aluno de forma similar sem levar em consideração o desempenho do mesmo além de que a ajuda é normalmente pré-definida e com pouca ou nenhuma customização (e.i., adaptação ao perfil do aluno) (WOOLF, 2009). Entretanto, pesquisas no campo da ciência cognitiva apontam que o processo de aprendizagem é influenciado por diferenças individuais e preferencias de estilos de aprendizagem (COUNCIL, 2000). Assim os sistemas educacionais tradicionais tendem a ter um impacto limitado, pois falham em se aproximar de uma abordagem "um-para-um"utilizada por um tutor humano que normalmente é oportunista, mudando dinamicamente os tópicos e métodos de ensino com base no progresso e desempenho de seus alunos (WOOLF, 2009).

Como resultado de pesquisas da comunidade de Inteligência Artificial (IA) aplicada à Educação, surge uma proposta promissora de suporte às atividades dos professores chamada de Sistemas Tutores Inteligentes (STIs), estes sistemas tem como propósito central o ensino individualizado e o uso de processos automatizados que buscam atender às dúvidas dos alunos no momento em que as mesmas surgem, contribuindo com a melhoria do processo de aprendizagem

(BLOOM, 1984). O uso de STIs pode trazer, também, benefícios relacionados a escalabilidade (i.e, possibilitam atender um maior número de alunos ao mesmo tempo), redução de custos, restrições físicas e temporais (WOOLF, 2009).

Devido aos benefícios citados, pode-se vislumbrar um grande potencial dos STIs, principalmente nos cursos de Educação a Distância (EAD) onde os STIs podem apoiar o professor através da automatização de parte das atividades pedagógicas, e.g., no auxílio em dúvidas mais recorrentes e na disponibilização de novas formas de aprendizagem (SILVA, 2011).

No Brasil, o censo da educação superior contabilizou 7,3 milhões de alunos matriculados em cursos de nível superior em 2013 sendo mais de 1,2 milhões matriculados em cursos a distância, o que equivale a mais de 15% das matriculas de graduação no país (INEP, 2013). Esses alunos estão matriculados em milhares de disciplinas em seus respectivos cursos e cada curso pode demandar um ou mais STIs o que indica que há a necessidade de se produzir STIs em larga escala para suprir a demanda.

Dada a grande necessidade de STIs, um questionamento que pode ser levantado é por que então ainda não existem milhares de STIs em vários domínios disponíveis para os professores. Uma resposta é que construir STIs ainda é uma tarefa complexa, principalmente do ponto de vista computacional, normalmente requer uma colaboração multidisciplinar entre engenheiros de software, professores e especialistas do domínio demandando aproximadamente 200 horas para cada hora de instrução, além disto, existem poucas ferramentas que facilitam esta tarefa (WOOLF, 2009).

A dificuldade de desenvolvimento de STIs é agravada por problemas encontrados na maioria das metodologias utilizadas na construção de STIs, tais como, as aplicações são desenvolvidas de forma independente, o conhecimento é normalmente codificado diretamente nas aplicações individuais, existe pouco reuso dos componentes do STI (i.e, modelo do estudante, modelo tutor, modelo do domínio e interface), necessitam de um linguagem padrão de representação de conhecimento e um conjunto de ferramentas que permita a manipulação do conhecimento, como consequência obtêm-se altos custos de desenvolvimento e manutenção do STI, falha na capacidade de interoperabilidade (i.e, capacidade do STI trabalhar em conjunto com outros STIs), requisitos muito restritivos da plataforma de implantação do STI, dificuldades de compartilhamento de material e de conhecimento do STI (SELF, 1999; RODRIGUES et al., 2005; SOTTILARE et al., 2015).

Em busca de reduzir as dificuldades de reuso e produção em larga escala de STIs sem comprometer a capacidade de customização (i.e., personalização) destes sistemas para os seus

usuários, a utilização de uma abordagem baseada em Linha de Produtos de Software (LPS) mostra-se com uma alternativa promissora, pois as LPSs permitem realizar a produção em larga escala de produtos de software de forma customizada, isto é possível por que uma LPS é um conjunto de sistemas de software que compartilham um conjunto comum e gerenciado de funcionalidades (chamadas de *features*) que satisfazem as necessidades específicas de um domínio ou seguimento de mercado e a partir deste núcleo comum de funcionalidades são desenvolvidos os softwares que atendem as necessidades especificas dos clientes (SEI, 2015; CLEMENTS; NORTHROP, 2001).

Mesmo utilizando uma abordagem de LPS para produzir STIs em larga escala e de forma customizada, há dificuldade de produzir estes sistemas de forma que seja possível o compartilhamento de conhecimento, a automatização do processo de construção e reconfiguração destas aplicações (GIRARDI; LEITE, 2008; RUSK; GASEVIC, 2008).

As ontologias são mecanismos formais para representação do conhecimento em diferentes domínios de maneira "compreensível"ao ser humano e ao computador e têm por objetivos permitir o compartilhamento e a reutilização do conhecimento (GRUBER, 2005; BORST, 1997). As ontologias podem ser utilizadas para possibilitar o compartilhamento do conhecimento das etapas de desenvolvimento da LPS, a automatização do processo de construção e reconfiguração de aplicações da LPS, o que pode ser utilizado para permitir contextualizar os recursos do STI às características de cada domínio (GIRARDI; LEITE, 2008; RUSK; GASEVIC, 2008).

Neste sentido, o trabalho de (SILVA, 2011) propõe uma LPS para STIs baseada no uso de tecnologias semânticas (e.g., ontologias) em busca de viabilizar a construção de STIs em larga escala, permitir que tal construção possa se adaptar aos diferentes domínios de conhecimento e permitir a evolução automatizada destes STIs. Esta abordagem utiliza agentes de software para ler as ontologias que descrevem a configuração da LPS e do STI e assim possibilitar a autoconfiguração dos mesmos de acordo com as novas funcionalidades adicionadas.

Na abordagem de LPS proposta por (SILVA, 2011) há duas fases: a Engenharia de Domínio (ED) e a Engenharia de Aplicação (EA). Na primeira etapa (ED) são construídos os componentes reutilizáveis que representam o domínio da LPS de STIs e na segunda etapa (EA) os componentes definidos na primeira etapa são reutilizados para instanciar (i.e., construir) um STI específico da LPS. Porém, na fase de EA da LPS se faz necessária a manipulação de ontologias e artefatos de software distintos, demandando a presença de profissionais com conhecimentos de ontologia e engenharia de software, o que dificulta a realização desta tarefa por autores/projetistas.

Este trabalho concentra-se na problemática de automatizar a fase de EA de uma abordagem de LPS baseada no uso de tecnologias semânticas. De forma que auxilie os autores/projetistas pelas tarefas da EA e reduza o esforço e a complexidade inerente as tecnologias utilizadas no processo.

1.2 OBJETIVOS

O objetivo geral desta dissertação é propor uma solução viável para auxiliar autores na engenharia de aplicação de LPS baseada no uso de tecnologias semânticas para construção de STIs, Este objetivo desmembra-se nos seguintes objetivos específicos:

Objetivo 1 Propor um modelo que torne o processo de engenharia de aplicação de uma abordagem de LPS baseada em tecnologias semânticas para construção de STIs semiautomático;

Objetivo 2 Implementar uma ferramenta que concretize o modelo proposto;

Objetivo 3 Realizar um estudo de caso utilizando a ferramenta implementada com o objetivo de avaliar o modelo proposto.

1.3 ESTRUTURA DA DISSERTAÇÃO

O conteúdo desta dissertação está estruturado em quatro capítulos, incluindo esta introdução que corresponde ao primeiro capítulo, os demais correspondem aos apêndices:

Capítulo 2 - Fundamentação Teórica Apresenta os conhecimentos fundamentais ao entendimento deste trabalho;

Capítulo 3 - Trabalhos Relacionados Apresenta os trabalhos relacionados a este trabalho;

Capítulo 4 - Solução Proposta Apresenta o modelo proposto através da descrição de uma arquitetura para o processo de engenharia de aplicação de uma LPS para STIs, das ontologias, regras semânticas e tecnologias utilizadas;

Capítulo 5 - Estudo de Caso e Avaliação Apresenta o estudo de caso realizado com uma ferramenta que implementa o modelo proposto e os resultados de sua experimentação;

Capítulo 6 - Conclusões e Trabalhos Futuros Apresenta as reflexões finais sobre este trabalho, as contribuições e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo apresentar os principais conceitos que formam a base teórica necessária para o entendimento deste trabalho. Iniciando pela Seção 2.1 onde são apresentados os conceitos relacionados as linhas de produto de software (LPSs), seguida pela Seção 2.2 que aborda a temática das ontologias e por fim na Seção 2.3 são apresentados os conceitos relacionados aos sistemas tutores inteligentes (STIs) e de ferramentas de autora de STIs.

2.1 LINHAS DE PRODUTO DE SOFTWARE

As empresas intensificam cada vez mais o uso de software nos seus produtos não importando a sua complexidade ou tamanho, isto vem aumentando o interesse das empresas de todos os portes e áreas de atuação em relação a competitividade no desenvolvimento de software (LINDEN et al., 2007).

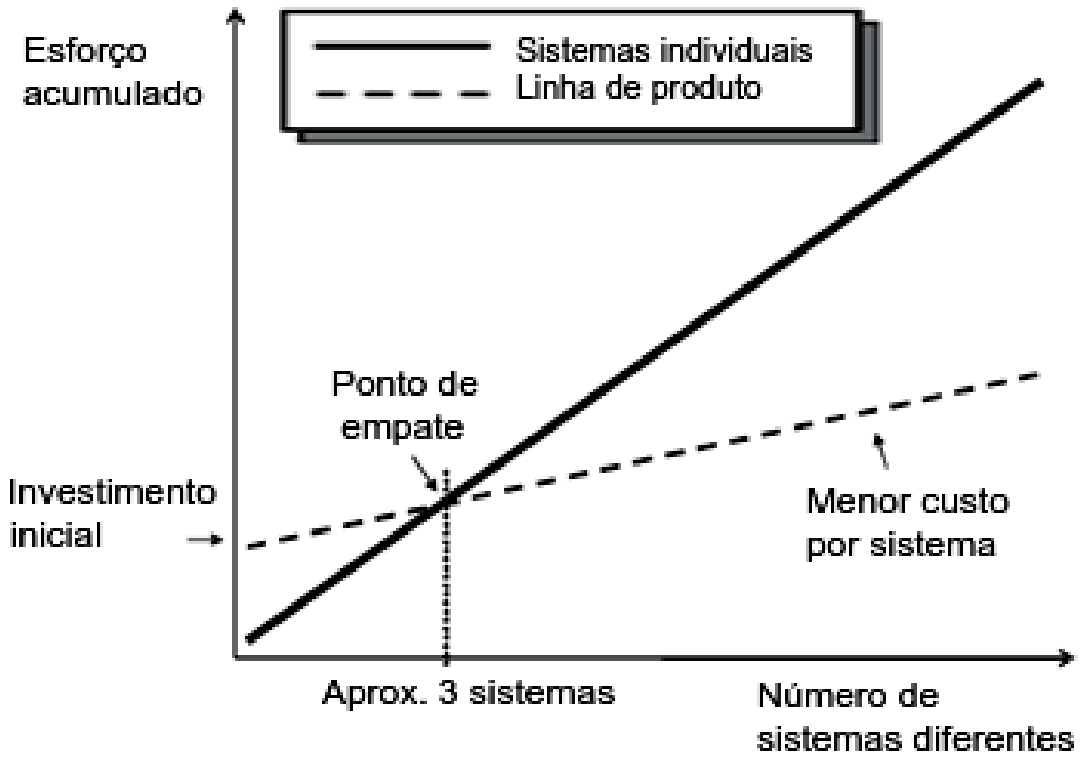
O desenvolvimento de software utilizando a engenharia de linha de produto de software (ELPS) vem sendo alvo de crescente interesse por parte das empresas. Entre as várias razões que levam as empresas a utilizarem a ELPS, a econômica é mais forte, uma vez que a abordagem de LPS dá suporte ao reuso em larga escala abrangendo grande parte dos processos do desenvolvimento de um software e de seus artefatos, permite a redução de custos, menor tempo de mercado (i.e tempo de entrega do produto ao cliente), maior facilidade de manutenção e aumento da qualidade do produto (LINDEN et al., 2007).

Diferente da abordagem tradicional de linha de produto utilizada na industria, que comumente realiza a produção em larga escala de produtos ao custo da diminuição da personalização dos mesmos, a linha de produto de software produz softwares em larga escala ao mesmo tempo que aumenta sua personalização de forma a entregar um produto mais adequado às necessidades de cada cliente. Em comparação com as abordagens tradicionais de reuso, a engenharia de linha de produto de software consegue reduzir tanto o custo de desenvolvimento do software quanto o tempo de mercado do mesmo. Outra diferença fundamental é que as abordagens tradicionais de reuso focam apenas no reuso dos artefatos de código enquanto que a ELPS busca reutilizar os artefatos essenciais de todas as fases do desenvolvimento do software, o que aumenta a qualidade dos produtos gerados devido as sucessivas revisões e correções realizadas em seus artefatos (POULIN, 1996).

Os benefícios em relação as abordagens de reuso tradicionais dependem, entretanto, de um investimento inicial maior que estima-se ser compensado após a geração de três produtos da

linha (WEISS; LAI, 1999). O gráfico apresentado na Figura 1 ilustra a relação entre o esforço acumulado na construção dos produtos (corresponde ao eixo vertical do gráfico) e o número de sistemas diferentes construídos (corresponde ao eixo horizontal do gráfico) sendo a reta contínua a representação do custo dos sistemas gerados utilizando a abordagem de reuso tradicional e a reta tracejada refere-se ao custo utilizando a abordagem da ELPS. No ponto de empate do gráfico ambas as abordagens de desenvolvimento possuem o mesmo custo na geração de um novo software, mas quando esse ponto é ultrapassado, a cada novo produto, o custo utilizando a abordagem de LPS diminui consideravelmente em relação ao custo de produção do software que utiliza a abordagem tradicional de desenvolvimento.

Figura 1 – Economia na utilização da ELPS em relação as abordagens de reuso tradicionais



Fonte: Adaptada pelo autor de (LINDEN et al., 2007).

2.1.1 Fundamentos

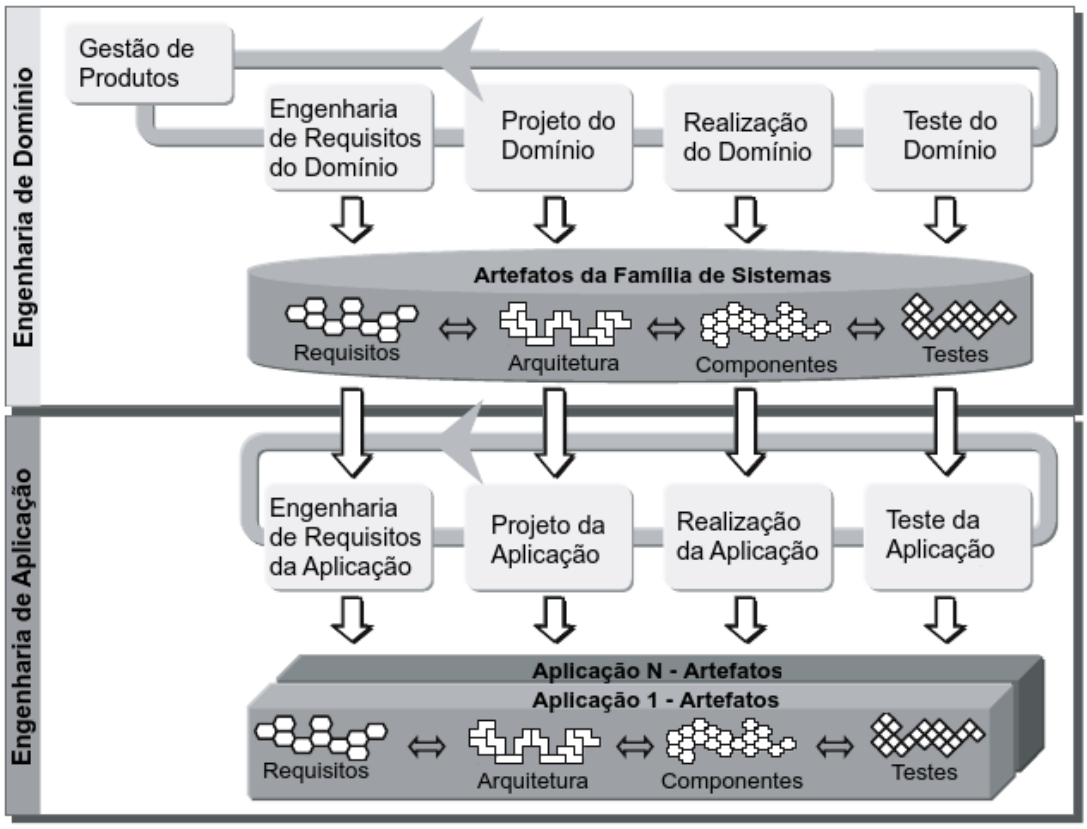
Uma linha de produto de software é uma família de softwares projetada de forma que possa produzir um conjunto de sistemas, similares entre si, que atenda as necessidades de uma missão ou segmento de mercado especifico (CLEMENTS; NORTHROP, 2001). Ou seja, esta abordagem diverge da abordagem de desenvolvimento de software tradicional, principalmente, por mudar o ponto de vista saindo do projeto de sistemas individuais para projetos que con-

templem uma família de sistemas que atendem a um seguimento de mercado (LINDEN et al., 2007).

Na construção de uma LPS deve-se utilizar uma modelagem e uma implementação que diferencie os pontos em comum e variáveis dos sistemas. Estes pontos formam um conjunto comum de ativos pré estabelecidos chamado de plataforma que serve como base para que novos produtos possam ser criados através da configuração deste conjunto comum de ativos, e.g., requisitos, modelo de domínio, arquitetura de software, documentação, planos de teste, conhecimento humano e habilidades, processos, componentes, serviços, entre outros. Os requisitos do domínio e a arquitetura da linha de produtos formam uma parte importante da plataforma, geralmente chamada de arquitetura de referência da LPS (LINDEN et al., 2007; POHL et al., 2005).

A engenharia de linha de produto de software é geralmente dividida em dois processos, engenharia de domínio e engenharia de aplicação (ilustrados na Figura 2), onde o primeiro foca na construção de uma plataforma robusta de linha de produto e o segundo na construção (i.e. instanciação) eficiente de aplicações individuais que atendam as demandas específicas dos clientes e do seguimento de mercado, com base na plataforma da LPS (METZGER; POHL, 2014). Os processos de engenharia de domínio e engenharia de aplicação serão apresentados, respectivamente, nas seções 2.1.3 e 2.1.4.

Figura 2 – Modelo de dois ciclos da Engenharia de Linha de Produto



Fonte: Adaptada pelo autor de (POHL et al., 2005).

2.1.2 Variabilidades nas LPS

Uma linha de produto de software deve dar suporte a uma conjunto de produtos de software que atendam as diferentes necessidades de seus clientes e podem atender diferentes segmentos de mercado. Para tanto a ELPS olha para a linha de produto como um todo e as possíveis variações entre seus produtos, sendo a variabilidade um propriedade essencial dos artefatos da fase de engenharia de domínio (LINDEN et al., 2007; POHL et al., 2005).

Variabilidades são as características que permitem à LPS ser customizada de forma a atender às necessidades de seus clientes, de forma geral, pode-se encontrar, em uma LPS, três tipos principais de características (POHL et al., 2005):

Comunalidade (i.e. semelhança) é uma característica comum a todos os produtos da linha de produto e deve ser implementada como parte da plataforma.

Variabilidade é uma característica comum a alguns produtos da linha e deve ser modelada

como uma possível variabilidade e implementada de forma que esteja disponível apenas nos produtos que a tenham selecionado.

Específicas do Produto é uma característica necessária para atender às necessidades de clientes individuais e que não faz sentido ser integrada à plataforma. Porém a plataforma deve ser capaz de suportar esta variabilidade. Enquanto as comunalidades e variabilidades são tratadas na engenharia de domínio, as características específicas do produto são tratadas apenas na engenharia de aplicação.

As características do sistema relevantes para o(s) *stakeholder(s)* (i.e. as partes interessadas no software) são chamadas de *features* e são utilizadas para modelar as comunalidades e variabilidades entre os sistemas da LPS. As *features* são definidas na engenharia de domínio e organizadas no modelo de *feature*[1] que representam as características de uma família de sistemas em um domínio e a relação entre elas (CZARNECKI et al., 2005).

A primeira notação utilizada para representar o modelo de *features* é a *Feature Oriented Domain Analysis* (FODA) apresentada por Kang et al. (1990) sendo também a mais largamente utilizada. Essa notação utiliza um diagrama em forma de árvore para representar o modelo de *features* e as associações entre as features que podem ser de um dos seguintes tipos:

Obrigatória (*Mandatory*) é uma *feature* que deve estar presente em todos os produtos da LPS (e.g., as *features* Chassi, Motor e Transmissão do modelo de *features* apresentado na Figura 3).

Opcional (*Optional*) é uma *feature* que pode estar presente ou não nos produtos da linha (e.g., a *feature* ArCondicionado do modelo de *feature* apresentado na Figura 3).

Alternativa (*Alternative*) indica que apenas uma *feature* de um conjunto de *features* pode estar presente no produto instanciado da LPS (e.g., as *features* Manual e Automática do modelo de *features* apresentado na Figura 3).

Or-Feature é semelhante à *feature* alternativa, define um conjunto de *features* onde apenas um subgrupo destas pode estar presente no produto (e.g., as *features* Gasolina e Álcool do modelo de *features* apresentado na Figura 3).

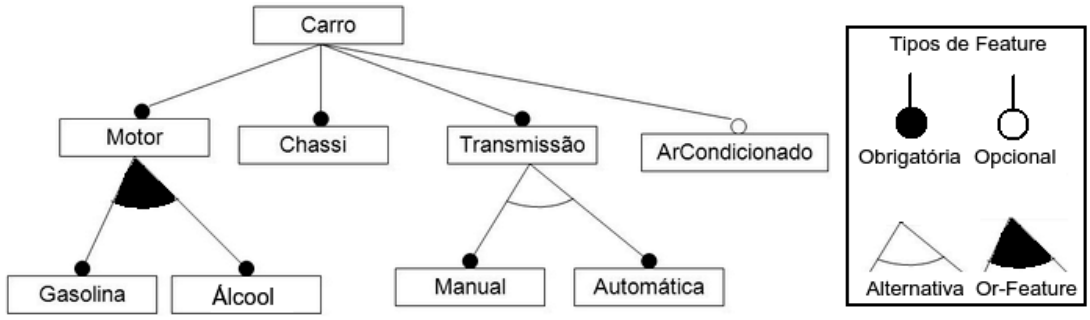
Além das associações estruturais das *features* também pode-se estabelecer regras de composição que modelam as dependências entre as mesmas, sendo que estas podem ser de dois tipos:

[1] Em inglês *Feature Model*

Requer (*Requires*) indica que uma *feature* quando selecionada requer a seleção de outra. Por exemplo, no modelo de *features* da Figura 3, caso existisse uma regra de composição: "ArCondicionado requer Automática" significa que selecionar a *feature* ArCondicionado implica selecionar a *feature* Automática.

Exclui (*Excludes*) uma *feature* quando selecionada elimina a possibilidade de selecionar uma outra. Por exemplo, no modelo de *features* da Figura 3, caso existisse uma regra de composição: "Manual exclui ArCondicionado" significaria que selecionar a *feature* Manual implica não poder selecionar a *feature* ArCondicionado.

Figura 3 – Modelo de *features* de um carro utilizando a notação FODA



Fonte: Autor desta dissertação, 2015.

O modelo de *features* é um documento de referência de grande importância que pode ser utilizado para vários propósitos no ciclo de desenvolvimento da linha de produto de software, além de fornecer a base para uma infra-estrutura destinada a reutilização melhorada, auxilia aos *stakeholders* a obter uma boa visão geral dos sistemas de uma linha de produto e aos analistas a gerenciar a complexidade de milhares de requisitos, desde que os mesmos estejam agrupados em *features* (KANG VIJAYAN SUGUMARAN, 2010).

Apesar dos modelos de *features* auxiliarem na gerência da complexidade inerente às LPS eles também tornam-se complexos ao ponto de ser inevitável o uso de uma ferramenta que os gerencie e conecte-os com seus requisitos o que facilita o desenvolvimento e evitar erros que custem muito caro (MASSEN; LICHTER, 2004; DHUNGANA et al., 2007). Logo, a utilização de ferramentas de suporte é essencial para manter a consistência do modelo de *features* e para realizar a instanciação correta dos produtos da linha (KANG VIJAYAN SUGUMARAN, 2010).

2.1.3 Engenharia de Domínio

A engenharia de domínio tem por objetivos definir as semelhanças e as variabilidades da LPS, o conjunto de aplicações que está prevista para a linha de produtos e construir os artefatos reutilizáveis que implementarão as variabilidades da LPS (POHL et al., 2005). O processo de engenharia de domínio é composto por cinco sub-processos, ilustrados na metade superior da Figura 2 e descritos a seguir:

Gestão de Produtos envolve as atividades de definição do escopo da LPS (i.e. a definição do conjunto de aplicações oferecidas pela linha de produtos), definição de quais *features* e quais artefatos de domínio são economicamente viáveis de serem reusados. Sendo a definição do escopo da LPS a principal atividade da gestão de produtos e geralmente envolve a presença de especialistas técnicos e de negócio (METZGER; POHL, 2014; HELFERICH et al., 2006; JOHN; EISENBARTH, 2009; GILLAIN et al., 2012).

Engenharia de requisitos do domínio envolve as atividades de negociação, documentação, elicitação, validação e gerenciamento de requisitos para o conjunto de produtos definidos na gestão de produtos. Para definir todos os requisitos relevantes a linha de produtos é necessário envolver um grande número de *stakeholders* e considerar fontes e restrições adicionais de requisitos (POHL et al., 2005). A quantidade de requisitos comuns e variáveis definidos na engenharia de requisitos do domínio tem um grande impacto em todas as outras atividades da ELPS.

Projeto do domínio envolve as atividades de definição da arquitetura de referência da LPS onde para cada decisão adiada é definida como um ponto de variação e um conjunto de alternativas (i.e. variantes). Para implementar a arquitetura de referência, tradicionalmente vem sendo utilizada uma abordagem baseada em componentes onde cada variabilidade é realizada através da composição de componentes ou introduzindo variações nos próprios componentes (JANOTA; BOTTERWECK, 2008). Mais recentemente, as abordagens consideradas pela comunidade são: as arquiteturas orientadas a aspecto que buscam melhorar a implementação de *features* transversais (i.e. que afetam outras *features*) (FIGUEIREDO et al., 2008) e as arquiteturas orientadas a serviço onde um serviço representa uma funcionalidade associada a uma característica de qualidade oferecida por um fornecedor de serviços através de uma interface de serviço (NITTO et al., 2008).

Realização do domínio envolve as atividades de detalhamento do projeto e implementação dos artefatos do domínio (e.g., componentes ou serviços reusáveis). As variabilidades podem ser implementadas utilizando linguagens de programação, compiladores e link editores (CAPILLA et al., 2013) e utilizar uma das abordagens existentes tais como: herança, programação orientada à aspectos, compilação condicional e substituição de código binário (METZGER; POHL, 2014).

Teste do domínio a realização de testes na ELPS tem por objetivo executar o software em busca de descobrir evidências de defeitos em artefatos do domínio antes que estes sejam reusados na engenharia de aplicação. Porém, devido às variabilidades nos artefatos do domínio, testar todas as potenciais aplicações da LPS durante a engenharia de domínio é impossível (POHL; METZGER, 2006). Logo, são utilizadas estratégias que buscam reduzir o número de artefatos a serem testados tais como: *pair-wise* (COHEN et al., 2008), *t-wise* (PERROUIN et al., 2012) ou focando nas *features* importantes (JOHANSEN et al., 2012). Esta atividade é essencial para que a ELPS seja bem sucedida pois uma falha em um artefato do domínio pode afetar todas as aplicações da linha de produto na qual este artefato foi reusado.

Os artefatos definidos na engenharia de domínio serão reusados na engenharia de aplicação para realizar a instanciação de um conjunto de aplicações da LPS.

2.1.4 Engenharia de Aplicação

A engenharia de aplicação tem por objetivos reutilizar os artefatos de domínio através da exploração de suas semelhanças e variabilidades, relacionar os artefatos com as necessidades específicas do cliente obedecendo as regras de derivação estabelecidas, possibilitando assim, a correta definição e instanciação de uma aplicação concreta da LPS (POHL et al., 2005). O processo de engenharia de aplicação é composto por quatro sub-processos, ilustrados na metade inferior da Figura 2 e descritos a seguir:

Engenharia de requisitos da aplicação envolve as atividades referentes à criação da especificação dos requisitos da aplicação. Tem como entrada os requisitos de domínio e uma relação com os principais recursos da aplicação correspondente. Também pode-se adicionar requisitos específicos da aplicação que não foram especificados na engenharia de domínio. Ao final deste sub-processo tem-se como saída a especificação dos requisitos para uma aplicação em particular (POHL et al., 2005).

Projeto da aplicação envolve as atividades de criação da arquitetura da aplicação específica. Utiliza a arquitetura de referência para instanciar a arquitetura da aplicação através da seleção e configuração das partes requisitadas da arquitetura de referência e a adição das características específicas da aplicação. Esse sub-processo tem como entradas a arquitetura de referência e a especificação dos requisitos da aplicação e obtêm como saída a arquitetura da aplicação específica (POHL et al., 2005).

Realização da aplicação envolve as atividades de instanciação da aplicação propriamente dita através da seleção e configuração dos componentes reusáveis de software bem como realização dos componentes específicos da aplicação. Este sub-processo tem como entradas a arquitetura específica da aplicação e os artefatos reusáveis da plataforma e a saída consiste na aplicação executável juntamente com os artefatos do projeto da aplicação (POHL et al., 2005).

Teste da aplicação contempla as atividades de verificação e validação de uma aplicação em relação a sua especificação. Este sub-processo tem como entradas todos os artefatos da aplicação, a aplicação instanciada e os artefatos reusáveis do teste do domínio e no final obtêm-se um relatório com os resultados dos testes realizados (POHL et al., 2005).

Ao final da execução dos sub-processos da engenharia de aplicação obtêm-se uma aplicação derivada da LPS.

2.2 ONTOLOGIAS

Na filosofia, comumente o significado de ontologia está associado ao "estudo do ser" ou mais especificamente a uma parte da filosofia que estuda o ser enquanto ser, concebido como tendo uma natureza comum inerente a todos e a cada um dos seres (HOLANDA, 1997). Na computação, mais especificamente na área de inteligência artificial, o termo ontologia é empregado para descrever uma especificação formal e explícita de uma conceitualização compartilhada (GRUBER, 1995). Por conceitualização pode-se entender um modelo abstrato de algum fenômeno do mundo, compartilhada pois é consenso de todos ou de um grupo e por especificação formal e explícita, pode-se entender que uma ontologia (incluindo suas partes e restrições) deve ter os tipos de conceitos utilizados e as restrições à sua utilização definidos explicitamente e de modo que possa ser manipulável por computador (CALERO et al., 2010).

2.2.1 Fundamentos

Uma ontologia pode ter uma variedade de formas, mas necessariamente deve incluir um vocabulário de termos e alguma especificação dos seus significados incluindo definições e indicações de como os termos estão inter-relacionados estabelecendo uma estrutura sobre o domínio e restringindo as possíveis interpretações dos termos (JASPER et al., 1999). Ou seja, além de uma ontologia definir um termo deve definir também a relação deste com outros termos, isto possibilita inferir conhecimento além da definição explícita dos termos.

De acordo com o grau de formalismo, aplicação, conteúdo ou função pode-se classificar as ontologias em tipos que adéquam-se a diferentes propósitos (ALMEIDA M; BAX, 2003), mais especificamente, de acordo com a sua função as ontologias podem ser classificadas nas seguintes categorias (GUARINO, 1997a; GUARINO, 1997b; GUIZZARDI, 2000):

Ontologia de Nível Superior descreve conceitos muito gerais, tais como: tempo, espaço, evento, processo e ação. Uma ontologia de nível superior é independente de domínio ou problema específico, possibilitando ser compartilhada por uma vasta base de usuários. Estes conceitos gerais podem ser especificados nas ontologias de domínio e tarefa.

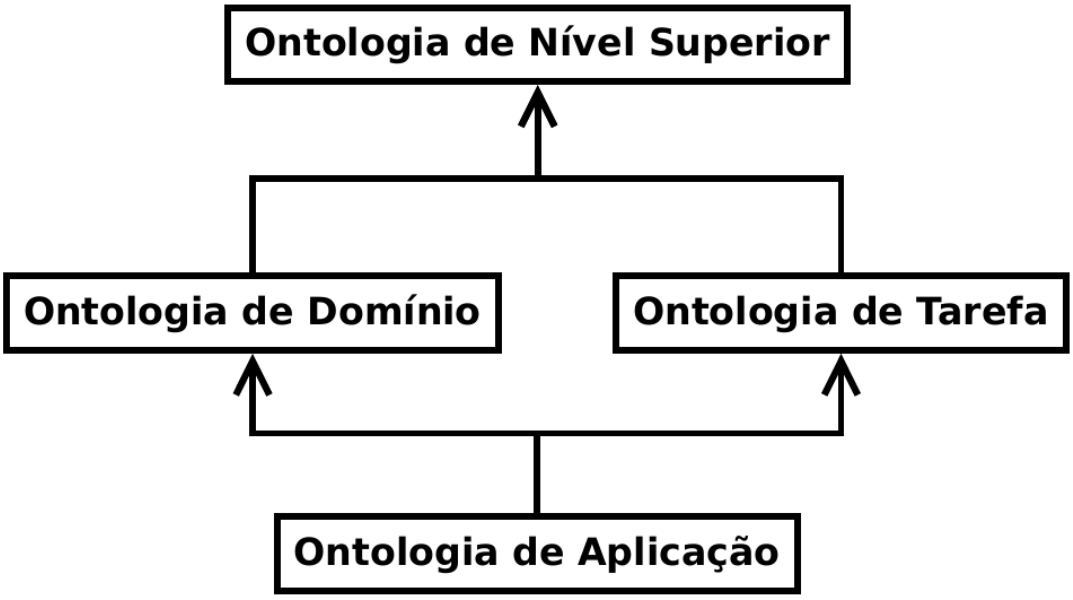
Ontologia de Domínio é o tipo de ontologia mais comum utilizada para definir vocabulários sobre conceitos e relacionamentos em domínios particulares, como por exemplo inteligência artificial e educação, a partir de termos de ontologias de nível superior.

Ontologia de Tarefa define vocabulários sobre conceitos relacionados a um tarefa genérica independente do domínio a qual será empregada, como por exemplo os processos de venda e compra, a partir de termos de ontologias de nível superior. Assim ontologias de tarefa podem ser utilizadas na resolução de problemas de forma independente do domínio.

Ontologia de Aplicação contém todos os conceitos necessários, tanto da tarefa quanto do domínio especifico, para modelar uma aplicação particular. Comumente, estes conceitos correspondem a regras aplicadas a um domínio particular quando alguma tarefa específica está sendo realizada.

A Figura 4 apresenta os níveis de dependência entre os tipos de ontologia descritos. No primeiro nível está a ontologia de nível superior que não depende de nenhuma outra, no segundo nível as ontologias de domínio e tarefa que dependem apenas da de nível superior e no terceiro nível a ontologia de aplicação que depende da de domínio e de tarefa.

Figura 4 – Diagrama dos níveis de dependência entre os tipos de ontologia segundo a classificação de (GUARINO, 1997a)



Fonte: Autor desta dissertação, 2015.

As ontologias podem ser classificadas, também, em *lightweight* e *heavyweight* (GOMEZ-PEREZ et al., 2003), a primeira refere-se as ontologias que descrevem taxonomias que incluem apenas definições, relações e propriedades de conceitos e a segunda classificação além de possuir o que a primeira possui, agrega axiomas e restrições aos conceitos, isto permite as ontologias *heavyweight* fornecer mais semântica e adicionar mais restrições em relação as *lightweight* (LEUNG, 2012).

De modo geral as ontologias possuem semelhanças em relação aos elementos que compõem a sua estrutura:

Classes são abstrações de conceitos do domínio da ontologia aceitos pela maioria, organizados em estruturas hierárquicas (i.e. taxonomias) utilizando o mecanismo de herança, como ilustrado na Figura 5 (CALERO et al., 2010).

Relações são representações das interações entre as classes de um domínio. Comumente as relações entre as classes de uma ontologias são binárias onde o primeiro argumento é chamado de domínio (ou *domain*) da relação e o segundo de alcance (ou *range*) da relação. As relações também são utilizadas para expressar o conceito de atributo (ou propriedade) que é uma relação cujo alcance é um tipo de dado, tal como um número, texto, etc.,

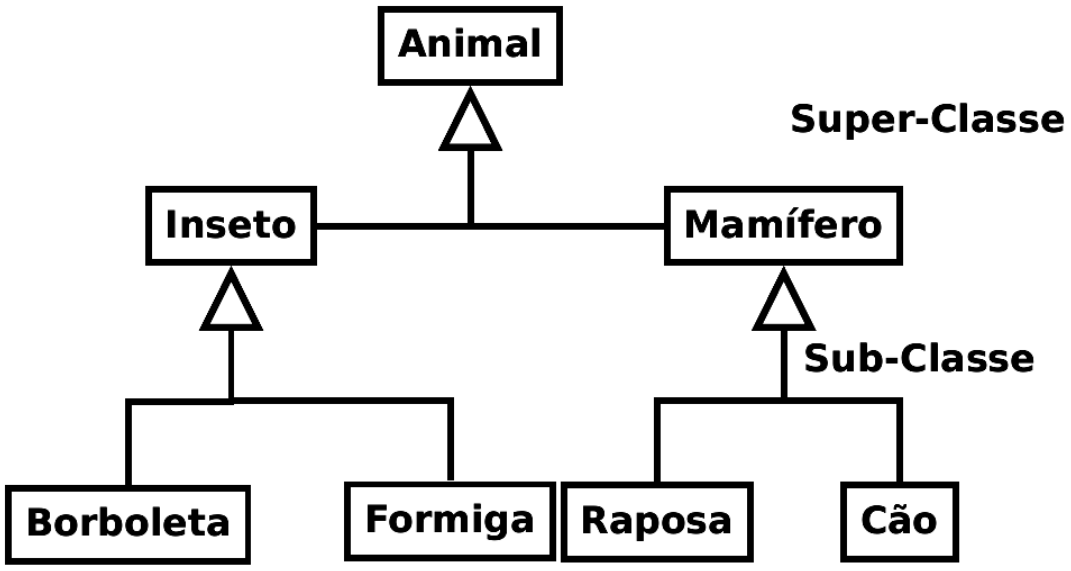
enquanto que nas relações o alcance é uma classe (CALERO et al., 2010).

Axiomas modelam sentenças que sempre são verdadeiras (ALMEIDA M; BAX, 2003; GRUBER, 1995). São utilizados para expressar conhecimentos que não podem ser expressos através de outros elementos sendo úteis para verificar a consistência da própria ontologia ou da sua base de conhecimento, também são úteis para inferir novos conhecimentos (CALERO et al., 2010). Um exemplo de axioma em um domínio de viagem é: "não é possível viajar do continente americano para o africano de trem".

Instâncias representam os elementos ou indivíduos em uma ontologia (CALERO et al., 2010). Um exemplo é "uma raposa vermelha sob registro 0002 com data de entrada no zoológico de pernambuco 21/12/2000".

Funções modelam eventos que podem acontecer no contexto da ontologia.

Figura 5 – Exemplo de uma hierarquia (i.e. taxonomia) de classes



Fonte: Autor desta dissertação, 2015.

2.2.2 Desenvolvimento de Ontologias

O objetivo principal da construção de ontologias é permitir o compartilhamento e reutilização de conhecimento. Para tanto, as ontologias devem ser desenvolvidas de forma que possam identificar o contexto de um termo, compartilhar suas definições e prover o suporte ao reuso, possibilitando assim auxiliar as pessoas na busca, extração, interpretação e processamento da

informação (BORST, 1997; GRUBER, 1993). Por este motivo, ontologias são utilizadas em uma grande variedade de aplicações, principalmente nas áreas de Engenharia do Conhecimento, Inteligência Artificial, Ciência da Computação, aplicações relacionadas a gerência do conhecimento, processamento de linguagem natural, comércio eletrônico, integração inteligente da informação, recuperação da informação, design e integração de banco de dados, bioinformática, educação e em novos campos emergentes, como a Web Semântica (GóMEZ-PéREZ MARIANO FERNáNDEZ-LóPEZ, 2004).

Para seja possível desenvolver uma ontologia é necessário primeiramente a definição do domínio e do escopo da mesma para que então seja definida (ou escolhida) um metodologia de desenvolvimento e uma linguagem que torne possível especifica-la formalmente.

Várias metodologias de desenvolvimento foram propostas com a intenção de guiar de forma sistemática a construção e manipulação de ontologias (ALMEIDA M; BAX, 2003). A seguir são descritas algumas das principais metodologias encontras na literatura:

Enterprise esta metodologia baseia-se em experiências no domínio empresarial, tem um ciclo de desenvolvimento dividido em quatro fases: identificação do propósito, identificação do escopo, formalização e documentação formal. A primeira fase, identificação do propósito, define o nível do formalismo de descrição da ontologia; a segunda fase, identificação do escopo, especifica o domínio no qual a ontologia deve modelar; a terceira fase, formalização, consiste na codificação, criação de formalismos e axiomas da ontologia; a quarta fase, documentação formal, gera a documentação da ontologia e realiza a revisão das fases anteriores (JONES et al., 1998).

Methontology esta metodologia fundamenta-se no desenvolvimento da ontologia a partir do conhecimento de um domínio e possui as seguintes atividades principais: especificação de requisitos, conceitualização do domínio do conhecimento, formalização do modelo conceitual em uma linguagem formal, implementação de um modelo formal e manutenção de ontologias implementadas. Durante o processo de desenvolvimento da ontologia, algumas atividades de suporte são realizadas, como: aquisição do conhecimento, integração, avaliação, documentação e gerenciamento de configuração (JONES et al., 1998).

On-To-Knowledge possui um ciclo de desenvolvimento composto por quatro fases: começo, refinamento, avaliação e manutenção. Na primeira fase, começo, é realizada a captura e especificação dos requisitos para o desenvolvimento da ontologia, são identificadas as questões de competência, são estudadas possíveis ontologias potencialmente reutilizáveis

> e uma primeira versão da ontologia é construída; na segunda fase, refinamento, uma versão melhorada é construída a partir da primeira; na terceira fase, avaliação, após a verificação dos requisitos e das questões de competência, a ontologia é implantada no ambiente de produção; na quarta fase, manutenção, é realizada a adequação da ontologia às mudanças de requisitos e as correções de erros (STAAB et al., 2001).

No desenvolvimento de ontologias, além da adoção de uma metodologia de desenvolvimento, é de grade importância utilizar ferramentas de construção de ontologia. O conjunto padrão de ferramentas deve incluir uma linguagem de representação de ontologia e um ambiente gráfico de desenvolvimento de ontologia (GAŠEVIĆ et al., 2006).

Para que se possa expressar as ontologias é necessário defini-las através de linguagens lógicas formais comumente chamadas de linguagens de ontologia. Essas linguagens tiveram seu inicio na década de 1990 quando foram criadas linguagens de ontologia baseadas em lógica de primeira ordem (e.g., KIF (GENESERETH et al., 1992)), em *frames* (e.g., Ontolingua (FARQUHAR et al., 1997), OCML (MOTTA, 1999) e FLogic (KIFER et al., 1995)) ou em lógicas de descrição (LD) (e.g., Loom (MACGREGOR, 1991)).

Com a popularização da Web, foram criadas linguagens de ontologia que exploravam as características deste ambiente, chamadas de linguagens de ontologia baseadas na Web ou linguagens de marcação de ontologias. Como exemplo desta categoria de linguagem destacam-se: SHOE (*Simple HTML Ontology Extensions*) é uma extensão da HTML (HEFLIN; HENDLER, 2000), XOL (*Ontology Exchange Language*) é fortemente baseada em XML (KARP et al., 1999), OIL (*Ontology Inference Layer*) (FENSEL et al., 2001) utiliza RDF(S) (MCBRIDE, 2004), DAML (*DARPA Agent Markup Language*) (HENDLER; MCGUINNESS, 2000). As duas últimas linguagens citadas foram combinadas na DAML + OIL (MCGUINNESS et al., 2002) (adicionou primitivas baseadas LD ao RDF(S)) e em 2001 a W3C (World Wide Web Consortium) formou um grupo de trabalho chamado de *Web-Ontology Working Group* com o objetivo de criar uma nova linguagem de marcação de ontologia chamada de OWL (*Ontology Web Language*) (CARDOSO; PINTO, 2015) como uma revisão da linguagem DAML + OIL. Entre tantas linguagens, RDF (*Resource Description Framework*), RDF Schema e OWL são as que vem sendo mais ativamente suportadas (CORCHO et al., 2003).

A W3C recomenda desde 2004 a utilização da linguem OWL como linguagem padrão para a representação de ontologias. Esta linguagem é utilizada para representar explicitamente o significado de termos em vocabulários e os relacionamentos entre os termos com possibilidade de alto nível de expressividade e inferência implícita, além disso, possui três dialetos (i.e.

sub-linguagens) incrementais que buscam atender às diferentes necessidades de seus usuários (MCGUINNESS et al., 2004):

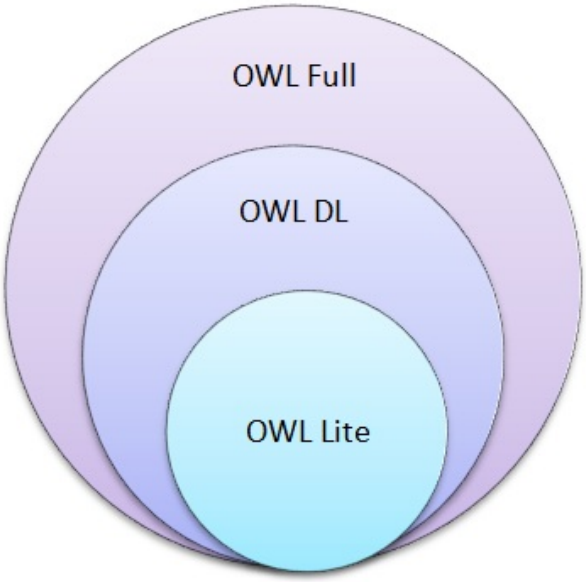
OWL *Lite* possui pouca expressividade, dá suporte apenas às necessidades primarias do usuário em relação a classificação em hierarquia e criação de restrições simples tais como cardinalidade e interseção de classes e restrições.

OWL DL dá suporte aos usuários que necessitam da máxima expressividade com decidibilidade, i.e., todos os cálculos devem terminar em um tempo finito e completude computacional, i.e., todas as conclusões são computáveis. Este dialetos corresponde a lógica descritiva e inclui todos os elementos da linguagem OWL apesar destes só poderem ser utilizados sobre certas restrições.

OWL *Full* possui o máximo de expressividade entre os dialetos da OWL o que fornece o suporte ao usuário a todos os elementos da linguagem OWL com a liberdade sintática do RDF com nenhuma garantia computacional.

Os dialetos OWL menos expressivos estão contidos dentro dos mais expressivos mas a recíproca não é verdadeira, ou seja, toda ontologia OWL *Lite* é uma ontologia válida em OWL DL e toda ontologia OWL DL é válida em OWL *Full*, como ilustrado no diagrama da Figura 6. Além disso, todo documento OWL é um documento RDF válido e todo documento RDF é um documento OWL *Full* válido, mas, nem todo documento RDF é um documento OWL *Lite* ou DL válido. Como exemplo da linguagem OWL, o Quadro 1 apresenta um trecho de uma ontologia descrita em OWL (MCGUINNESS et al., 2004).

Figura 6 – Diagrama de Venn dos dialetos OWL



Fonte: Disponível em: <http://semanticworld.altervista.org/blog/2011/05/owl-chi-e-e-cosa-fa/>. Acesso em: 04 ago. 2015.

Quadro 1 – Exemplo de um trecho de uma ontologia em OWL.

```
1  <owl:Class rdf:ID="DescritorVinho" />
2
3  <owl:Class rdf:ID="CorVinho">
4    <rdfs:subClassOf rdf:resource="#DescritorVinho" />
5  </owl:Class>
6
7  <owl:ObjectProperty rdf:ID="temDescritorVinho">
8    <rdfs:domain rdf:resource="#Vinho" />
9    <rdfs:range  rdf:resource="#DescritorVinho" />
10 </owl:ObjectProperty>
11
12 <owl:ObjectProperty rdf:ID="temCor">
13   <rdfs:subPropertyOf rdf:resource="#temDescritorVinho" />
14   <rdfs:range rdf:resource="#CorVinho" />
15 </owl:ObjectProperty>
```

Fonte: Autor desta dissertação, 2015.

Existe uma ampla variedade de ambientes gráficos de desenvolvimento de ontologia, e.g Ontolingua (GRUBER, 1992), WebODE (ARPÍREZ et al., 2001), Protégé (MUSEN, 2015) e OntoEdit (SURE et al., 2002). De modo geral estas ferramentas podem ser divididas em dois

grupos: as que dão suporte a uma linguagem de representação específica e as que proveem uma suíte de ferramentas integradas e extensíveis que em sua maioria são independentes de linguagem e fornecem funcionalidades além das relacionadas a simples edição de ontologias (GóMEZ-PéREZ MARIANO FERNáNDEZ-LóPEZ, 2004).

As ferramentas de desenvolvimento de ontologia modernas geralmente são pertencentes ao segundo grupo citado (suíte de ferramentas integradas e extensíveis) e possuem uma arquitetura em três camadas que possibilita a clara distinção entre as ontologias do *backend* (modelo) e as ontologias que armazenam o conhecimento, os módulos da lógica de negócio e os módulos da lógica da aplicação, a interface interna da ferramenta e as interfaces da aplicação com o usuário (GóMEZ-PéREZ MARIANO FERNáNDEZ-LóPEZ, 2004). A seguir são descritos alguns dos principais ambientes gráficos de desenvolvimento de ontologia modernos:

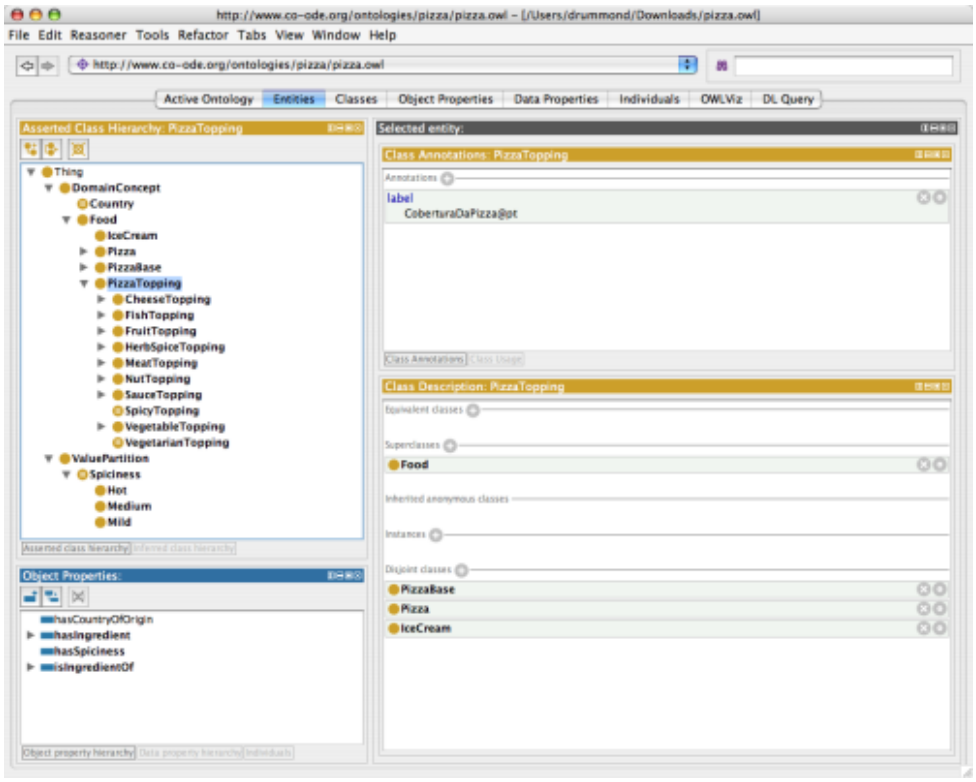
WebODE *Web Ontological Engineering Workbench* (WebODE) foi desenvolvido na Universidade Politécnica de Madri (UPM) [2]. É uma ambiente Web composto por um editor de ontologias, um gestor de conhecimento (ODEKM), um portal gerador automático de web semântica (ODESeW), uma ferramenta de anotação de recursos Web (ODEAnnotate) e uma ferramenta de edição de serviços web semântica (ODESWS) (FERNANDES, 2007). Outra característica deste ambiente é a persistência das ontologias em bases de dados relacionais. Fornece como principais funcionalidades: serviços de documentação, avaliação e fusão de ontologias, suporte para metodologia *METHONTOLOGY* e importa/exporta, entre outras linguagens, para XML, RDF(S), OIL, DAML + OIL, Flogic e Prolog (ARPÍREZ et al., 2001).

OntoEdit *Ontology Engineering Environment* (OntoEdit) foi desenvolvido pelo AIFB na universidade de Karlsruhe. É um ambiente de desenvolviemnto de ontologias semelhante ao WeODE com uma arquitetura baseada em *plugins*, extensível e flexível que provê como principais funcionalidades explorar e editar ontologias. Este ambiente pode incluir *plugins* que realizam: inferência utilizando OntoBroker (DECKER et al., 1999), edição gráfica de regras, importação e exportação de ontologias em vários formatos (FLogic, XML, RDF(S), DAML + OIL), entre outras funcionalidades. O OntoEdit é disponibilizado nas versões gratuita (OntoEdit Free) e paga (OntoEdit Professional); e uma suíte de ferramentas, KAON (Karlsruhe Ontology), foi desenvolvida como a sucessora do OntoEdit (SURE et al., 2002).

[2] <http://www.fi.upm.es/>

Protégé desenvolvido pela universidade de Stanford [3] é o ambiente de desenvolvimento de ontologia mais amplamente utilizado. Consiste em uma aplicação *standalone* java e possui uma arquitetura extensível. O centro desse ambiente é um editor de ontologia (mostrado na Figura 7) ao qual agrega uma biblioteca de *plugins* que adicionam mais funcionalidades ao ambiente. Existem *plugins* para importar e exportar a ontologia para diferentes linguagens (RDF(S), XML, XML Schema, OWL, Flogic, Jess, Prolog, OIL), projeto de linguagem de ontologia, acesso OKBC, criação e execução de restrições, junção de ontologias, entre outros. Este ambiente provê suporte as principais atividades do ciclo de desenvolvimento de ontologias, tais como: conceitualização, integração, avaliação, reuso, aquisição de conhecimento e gerenciamento da ontologia (CALERO et al., 2010).

Figura 7 – Tela do editor de OWL do ambiente de desenvolvimento de ontologias Protégé



Fonte: Disponível em: <http://protegewiki.stanford.edu/images/5/5c/Tab-entities.png>. Acesso em: 04 ago. 2015.

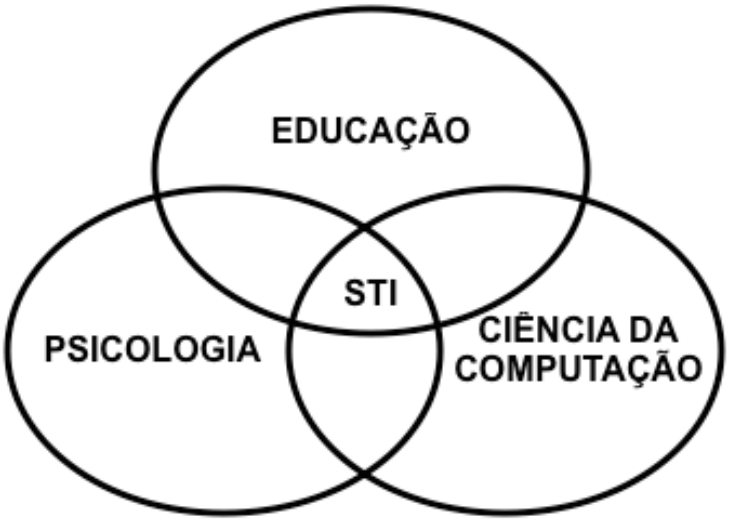
[3] <http://www.stanford.edu/>

2.3 SISTEMAS TUTORES INTELIGENTE

Sistemas Tutores Inteligentes são sistemas instrucionais baseados em computador com modelos de conteúdo instrucional que especificam “o que” ensinar e estratégias de ensino que especificam “como” ensinar (WENGER, 1987). Estes softwares dão suporte às atividades de aprendizagem de forma melhorada em relação aos sistemas tradicionais de instruções assistidas por computador (CAI - *computer aided instruction*) pois incorporam técnicas de Inteligência Artificial (IA) que buscam simular o processo do pensamento humano para auxiliar na resolução de problemas ou na tomada de decisão (GAMBOA; FRED, 2002; FOWLER, 1991). Eles têm a capacidade de auxiliar os estudantes no aprendizado da resolução de problemas complexos que necessitam de uma interface de interação humano-máquina muito flexível (BURNS et al., 2014) e têm como principal objetivo prover um ensino individualizado e adaptado a cada estudante.

Para que os STIs possam modelar “o que”, “como” e “a quem” ensinar é necessário um embasamento multidisciplinar entre a Ciência da Computação, Psicologia e Educação (como ilustrado na Figura 8) sendo que a Ciência da Computação, particularmente a área de Inteligência Artificial, trata como discutir sobre inteligência; a Psicologia, particularmente a Ciência Cognitiva, trata como as pessoas pensam e aprendem e a Educação em como apoiar o ensino de uma forma melhor (WOOLF, 2009).

Figura 8 – Diagrama de Venn com o posicionamento dos STIs entre a áreas: Ciência da Computação, Psicologia e Educação



Fonte: Autor desta dissertação, 2015

2.3.1 Arquitetura Tradicional

Existem várias arquiteturas para o desenvolvimento de sistemas tutores inteligentes, sendo que geralmente a arquitetura de um STI é composta pelos seguintes componentes: modelo do domínio (ou especialista), modelo do tutor (também chamado de instrucional ou pedagógico), modelo do aprendiz (ou estudante) e o módulo de interface entre o usuário humano e o computador (BURNS et al., 2014). A Figura 9, ilustra a arquitetura clássica de um STI mostrando os módulos citados e as setas indicam a interação entre eles.

Existem dois pontos principais que atribuem inteligência a um STI, o primeiro é o conhecimento que o sistema tem sobre o domínio a ser trabalhado e o segundo corresponde a quais são os princípios que ele utiliza para tutorar e os métodos pelos quais ele aplica estes princípios (POLSON; RICHARDSON, 2013). Na arquitetura clássica de STIs, o modelo do domínio é conceitualmente o primeiro modelo a ser construído, responsável por definir o conhecimento do domínio em que o tutor irá ensinar. Este modelo armazena todo o conhecimento sobre o que será ensinado e deve fornecer o conhecimento e a informação de forma flexível e adaptativa (SALDÍAS et al., 2002). Vale salientar que uma grande quantidade de esforço é aplicada neste modelo (em torno de 50%), pois, deve-se realizar a descoberta e codificação do conhecimento sobre o conteúdo específico (i.e. domínio) de forma declarativa e procedimental (POLSON; RICHARDSON, 2013; RODRIGUES et al., 2005).

O modelo do aprendiz é responsável por modelar os conhecimentos adquiridos pelo aprendiz e suas interações com o sistema de forma que o STI possa, a partir deste modelo, inferir automaticamente qual a estratégia pedagógica mais adequada ao aprendiz além de identificar suas preferências, possibilitando ao sistema oferecer uma experiência de interação personalizada.

A forma de implementação deste modulo depende dos objetivos e características do processo de ensino-aprendizagem utilizado. Vem sendo discutida uma proposta revolucionária, chamada de modelo aberto do estudante, onde o perfil do aprendiz acerca do domínio fica disponível para visualização, podendo ser feita tanto de forma simples, mostrando o progresso do aprendiz com um subconjunto de conhecimentos especializados, como de formas mais complexas, tais como uma estrutura hierárquica em árvore, um gráfico conceitual ou até mesmo descrições de conhecimento e ideias erradas (BULL; KAY, 2007).

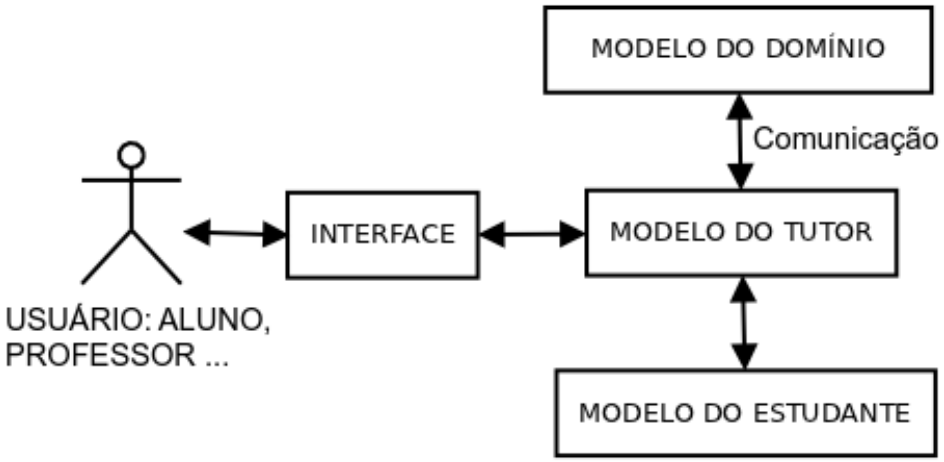
No modelo aberto do estudante, o aprendiz passa a ter um certo nível de responsabilidade sobre o conteúdo do seu modelo de aprendiz com possibilidade de modifica-lo diretamente ou através de negociação, onde o aprendiz registra sua discordância com a avaliação do sistema sobre seu nível de compreensão em um determinado tema e propõe uma mudança para um nível

diferente, o que pode acarretar em uma melhoria do modelo (BULL; MABBOTT, 2006; BULL; KAY, 2010).

O modelo do tutor tem todo o conhecimento do tutor de como ensinar pois reúne as estratégias e táticas de ensino e as aplica de acordo com o modelo do aprendiz e o modelo de domínio. Essas estratégias podem ser derivadas de observações empíricas dos professores informadas através das teorias de aprendizagem ou através de recursos computacionais apenas fracamente relacionados a um humano tais como simulações computacionais ou personagens animados (WOOLF, 2009).

O modelo da interface contêm os métodos que permitem a comunicação entre os estudantes e o computador (e.g interfaces gráficas, personagens animados e mecanismos diálogos), por tanto, é de grande importância que este modelo seja muito bem desenvolvido de forma que possibilite uma interação entre o usuário e o sistema o mais confortável e intuitiva possível (WOOLF, 2009). Este modelo tem entre suas principais responsabilidades apresentar os conteúdos do modelo do domínio e permitir que eles sejam explorados pelo aprendiz.

Figura 9 – Arquitetura clássica de um STI



Fonte: Autor desta dissertação, 2015.

2.3.2 Ferramentas de Autoria de STIs

De modo geral, o processo de autoria consiste em produzir ou organizar materiais que possam ser utilizados no processo de ensino-aprendizagem (e.g., STI, Objeto de Aprendizagem), sendo as ferramentas de autoria geralmente disponibilizadas como softwares que permitem a um autor (e.g., professor) criar, manipular, alterar e excluir estes materiais educacionais (MARCZAL, 2014).

Apesar da efetividade dos STIs (VANLEHN, 2011; KULIK; FLETCHER, 2015), sua adoção é limitada principalmente por causa de seu alto custo de desenvolvimento, capacidade de reuso limitada, necessidade de habilidades especializadas, falta de padrões (tem como consequência baixa interoperabilidade entre STIs) e capacidade inadequada de adaptação as necessidades dos aprendizes (SOTTILARE et al., 2015).

Devido as dificuldades inerentes à construção de STIs, é importante utilizar ferramentas de autoria para aumentar a adoção destes sistemas. Portanto, as ferramentas de autoria de STIs são objetos de várias pesquisas da comunidade de STIs (MURRAY, 1999; MURRAY, 2003; SOTTILARE; GILBERT, 2011; SOTTILARE et al., 2012; SOTTILARE, 2013; SOTTILARE et al., 2015). Nestas pesquisas, foram identificados vários objetivos associados aos processos de autoria de STIs podendo estes serem agrupados em quatro categorias: redução do esforço requerido do autor, redução do conhecimento requerido pelo autor, suporte à organização do conhecimento do domínio e possibilidade de rápida avaliação de protótipos (SOTTILARE et al., 2015).

Alguns exemplos de ferramentas de autoria de STIs bem sucedidas são: ASPIRE (MITROVIC et al., 2009), AutoTutor (BENJAMIN et al., 2014), CTAT (ALEVEN et al., 2015), GIFT (SOTTILARE, 2013) e SitPed (LANE et al., 2015). Mas ainda há barreiras que fazem com que as ferramentas de autoria de STIs existentes não sejam utilizadas pelo grande público, tais como: habilidades especializadas, e.g, programação de computadores e entendimento de design instrucional são algumas das principais habilidades requeridas dos autores; tempo e custo para construir STIs, isto é devido principalmente a alta complexidade dos STIs e as deficiências de usabilidade das ferramentas de autoria; demora para recuperar e organizar o conteúdo de autoria e falta de padrões para a autoria de STIs (SOTTILARE et al., 2015).

3 TRABALHOS RELACIONADOS

Este capítulo tem por objetivo apresentar os trabalhos relacionados a esta dissertação. Inciando pela Seção 3.1 que apresenta uma das ferramentas de autoria que mais se destaca na comunidade. Em seguida a Seção 3.2 apresenta alguns dos principais desafios enfrentados pelas ferramentas de autoria na construção de STIs. E por fim a Seção 3.3 apresenta a abordagem para construção de STIs baseada em linha de produto de software semânticas, ou seja, LPS que utilizam tecnologias semânticas (e.g., ontologias).

Exemplos de ferramentas de autoria de STIs bem sucedidas são apresentadas nas próximas sessões.

3.1 CTAT

No últimos 25 anos surgiram várias ferramentas de autoria de STIs mas poucas continuam evoluindo. Entre elas a que mais se destaca pelo seu sucesso é a *Cognitive Tutor Authoring Tools* (CTAT), desenvolvida pela universidade Carnegie Mellon, tem uma longa historia e permanecer viável até hoje (SOTTILARE et al., 2015).

O CTAT é um conjunto de ferramentas que busca permitir a construção de STIs que empregam a técnica da aprendizagem pela ação (ou aprender fazendo) para cursos online. Estes sistemas tutores podem ser do tipo *Cognitive Tutors* (i.e., tutores cognitivos) ou *Example-Tracing* (i.e., rastreamento de exemplo), descritos a seguir:

Cognitive Tutor este é o tipo de tutor mais robusto, fundamentado na teoria de aprendizagem e cognição ACT-R (ANDERSON, 2014), onde o comportamento de resolução de problemas do estudante é interpretado como um modelo cognitivo construído utilizando regras de produção que definem quais conhecimentos o estudante deve adquirir (KOEDINGER et al., 1997). O desenvolvimento destes tutores tem a desvantagem de demandar muito mais tempo e requerer programação de inteligencia artificial (IA), entretanto, o modelo criado funciona com uma gama de problemas e de forma mais flexível, uma vez que o tutor é capaz de reconhecer várias formas de solução dos estudantes e dependências entre as etapas de resolução (ALEVEN et al., 2006).

Example-Tracing Tutor este tipo de tutor possui as principais funcionalidades dos tutores cognitivos sem necessidade de programação. Estes tutores são criados através de "demonstração", ou seja, o autor demonstra ao sistema como o estudante deve resolver o problema

e quais os erros esperados (KOEDINGER et al., 2004). Assim, esses tutores tem como a principal vantagem não requerer programação de IA e como principal desvantagem demandar mais tarefas de autoria (e.g demonstrar e anotar) para cada problema de forma separada (ALEVEN et al., 2006).

O CTAT é composto por uma interface principal da ferramente e por duas ferramentas externas (ver Figura 10):

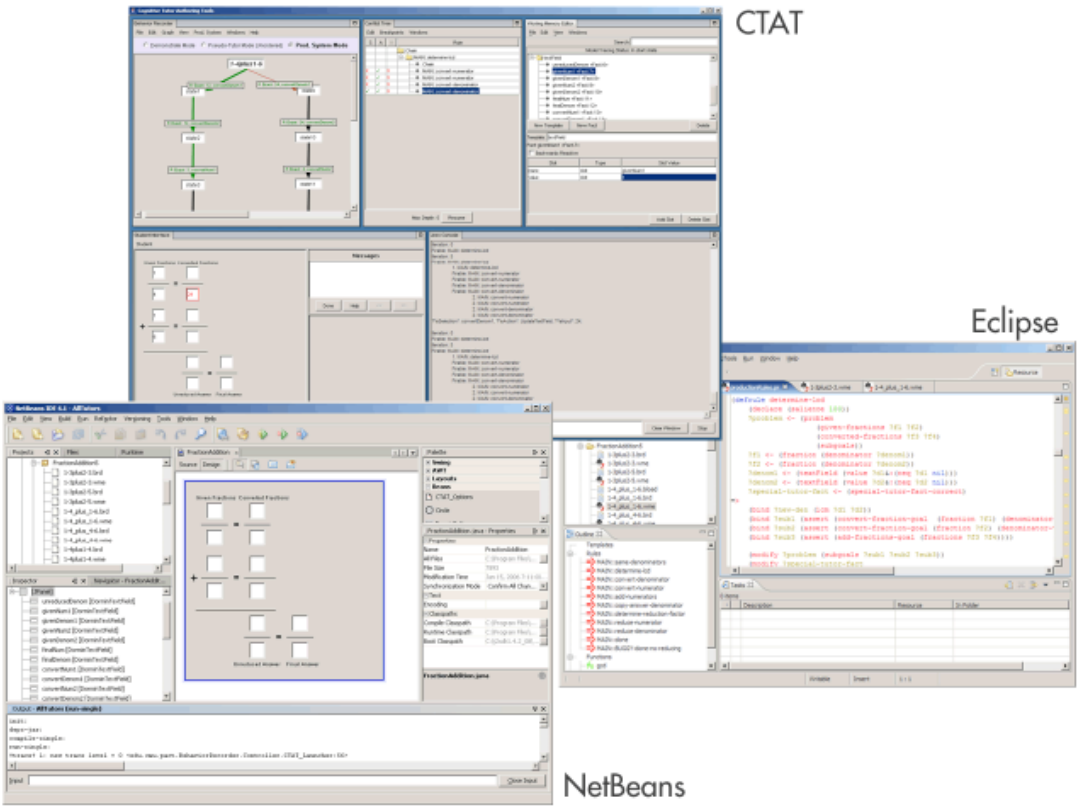
Construtor de Interface Gráfica é uma ferramenta externa (*NetBeans* ou *Macromedia Flash*) enriquecida com um conjunto de componentes de interface (*widgets*) "tutoráveis" desenvolvidos pelo CTAT para criar a interface gráfica dos problemas com o estudante. Este construtor suporta a criação da interface com o estudante sem programação, utilizando a técnica de *drag and drop*, onde basta que o autor arraste os *widgets* disponíveis para uma determinada área e organize-os (ALEVEN et al., 2006).

Registrador de Comportamento ferramenta central com três funções principais: registrar exemplos de comportamentos corretos e incorretos demonstrados pelo autor na interface do estudante em forma de um grafo de comportamento; implementar a função *Example-Tracing*; prover suporte ao planejamento e teste de modelos cognitivos (ALEVEN et al., 2006).

Editor Externo é utilizado para editar regras de produção *Java Expert System Shell* (JESS[1]) para o modelo cognitivo. O *plug-in* para a plataforma Eclipse que permite a verificação da sintaxe e a funcionalidade de auto completar das regras JESS (ALEVEN et al., 2006).

[1] <http://www.jessrules.com/>

Figura 10 – Ferramentas do CTAT



Fonte: (ALEVEN et al., 2006) .

3.2 DESAFIOS NA CONSTRUÇÃO DE STIS

Alguns dos principais desafios da engenharia de software na educação estão relacionados as variabilidades dos STIs e a construção destes sistemas em larga escala, como apresentado no artigo (BITTENCOURT et al., 2012).

Tomando como base o artigo (BITTENCOURT et al., 2012) alguns dos principais desafios relacionados as ferramentas de autoria são:

Reduzir o Conhecimento Prévio Requerido do Autor o processo de autoria de STIs envolve conhecimentos avançados de computação e sobre os módulos do STI. Assim, a construção de um STI é uma tarefa complexa que demanda capacitação dos aspirantes a autor de STIs. Logo, reduzir a quantidade de conhecimento prévio necessário (i.e., habilidade requerida) para que o autor construa STIs é importante, pois aumenta a quantidade de autores habilitados para esta tarefa e reduz os custos com treinamento.

Facilitar a Autoria de STIs a construção de um STI é uma tarefa complexa e custosa tanto do ponto de vista do esforço quanto do tempo empregado. Assim, é importante que as abordagens de construção dos STIs disponibilizem mecanismos que facilitem as tarefas de autoria destes sistemas.

Permitir o Reuso e Compartilhamento de Conhecimento Independente do Domínio devido a grande variabilidade dos domínios e subdomínios de conhecimento presentes na educação, principalmente no nível superior, e dado que os STIs são sistemas dependentes de domínio de conhecimento, é uma característica importante e desejável que as abordagens de construção de STIs possibilitem a reutilização e o compartilhamento dos conhecimentos gerados de forma independente de domínio (BITTENCOURT et al., 2012).

Permitir a Construção de Famílias de STIs devido a grande demanda existente para STIs e a grande variabilidade gerada pelos diferentes contextos em que os STIs são aplicados, é necessária a construção de famílias de STIs para atender a instanciação em larga escala destes sistemas de forma adequada a cada contexto.

Facilitar a Instanciação de STI em Larga Escala a etapa de instanciação de STIs a partir de uma família de produtos de software é realizada na fase de engenharia da aplicação geralmente de forma estática. Este processo comumente é uma tarefa complexa e demorada para autores que não são engenheiros de software. Assim, facilitar este processo permite aos autores que não tem conhecimento de engenharia de software realizar a instanciação dos STIs de acordo com cada domínio a ser trabalhado de forma mais simples e personalizada.

Rapidez de Instanciação de STIs em Larga Escala por Autores este ponto está relacionado ao ponto anterior e parte da ideia que a preparação do STI deve ser realizado por autores como professores de disciplinas em um curto espaço de tempo e estes estão envolvidos em várias atividades além da instanciação do STI (BITTENCOURT et al., 2012). Torna-se fundamental que o processo de instanciação de STIs seja o mais rápido possível.

Adaptar os Requisitos dos STIs ao Usuário para prover um ensino individualizado e maximizar o aprendizado do aluno, é importante ter não apenas um tutor inteligente adequado a cada domínio (requisitos estáticos), mas também um tutor que possa adequar as suas funcionalidade a cada estudante (requisitos dinâmicos) (BITTENCOURT et al., 2012).

Facilitar a Manutenção dos STIs comumente os sistemas de software necessitam de modificações depois de terem sido colocados em uso. Isto pode ser devido a necessidade de corrigir

defeitos de código, de projeto, de especificação ou melhorar o desempenho. Ao longo do tempo, o custo destas modificações tende superar o custo de desenvolvimento do software. Logo, facilitar a manutenção dos STIs criados traz benefícios como a redução de custos e a diminuição das chances que a mesma gere novos defeitos.

Permitir a Evolução dos STIs incluir novas funções e efetuar melhoramentos gerais é importante para manter o STI útil por mais tempo e mais adequado aos requisitos do usuário. Facilitar a evolução dos STIs e a manutenção dos mesmos traz redução de custos e a diminuição das chances que a manutenção gere novos defeitos.

3.3 LINHAS DE PRODUTO DE SOFTWARE SEMÂNTICAS

A abordagem de linha de produto de software (LPS) dá suporte ao reuso em larga escala, abrangendo grande parte dos processos de desenvolvimento de um software e de seus artefatos, permite a redução de custos, menor tempo de entrega do produto, mais facilidade na manutenção e aumento da qualidade do produto (LINDEN et al., 2007). Logo a utilização da abordagem de LPS é interessante para diversos domínios que necessitam produzir softwares em larga escala e com alto grau de personalização dos mesmos, como é o caso dos STIs.

Sob a perspectiva da Web Semântica, outra necessidade dos STIs é permitir que seus recursos educacionais sejam anotados semanticamente viabilizando uma configuração automatizada destes sistemas e que os mesmos possam ser distribuídos e compartilhados com mais autonomia (BITTENCOURT et al., 2008). Por isso, as ontologias vêm sendo gradativamente inseridas no processo de construção dos STIs para anotar semanticamente os recursos educacionais destes sistemas (JOVANOVIC et al., 2009; MERINO; KLOOS, 2008; BITTENCOURT et al., 2009). Mesmo em outros domínios, as ontologias podem ser úteis na anotação semântica dos componentes de software de forma que possam ser rastreados, distribuídos e compartilhados mais facilmente.

O trabalho de (SILVA, 2011) propõe uma abordagem de construção de STIs através da combinação de LPS e a web semântica (i.e., linha de produto de software semântica). Esta abordagem de construção de STIs permite produzir STIs em larga escala, de forma customizada, adaptados a diferentes domínios e de modo que tanto a LPS quanto o STI possam ser reconfigurados quando novas funcionalidades surgirem. Esta abordagem tem duas fases, a engenharia de domínio (ED) e a engenharia de aplicação (EA).

Na primeira fase (ED) são construídos os artefatos de software da LPS de STIs e anotados

utilizando ontologias para que seja possível haver um processo automatizado e dinâmico de construção, configuração e manutenção dos STIs da LPS. Também realiza-se a autoria do domínio, processo no qual são especificados os domínios de conhecimento de forma que possam ser utilizados pela LPS e seus produtos. A autoria do domínio é realizada seguindo os seguintes passos: Especificação da Hierarquia do Domínio, Cadastro de Conteúdo, Cadastro de Problemas, Ordenar Currículos, Ordenar Recursos Educacionais e Exportar Domínio (SILVA, 2011).

Na segunda fase (EA) são reutilizados os artefatos construídos na ED para realizar a construção um STI específico. Para tanto, realiza-se a configuração do STI através de ontologias que descrevem a LPS e as decisões do autor em relação a quais funcionalidades farão parte do STI, instanciação do STI com as funcionalidades definidas e implantação do STI em um servidor Web (SILVA, 2011). Vale salientar que a EA requer conhecimentos avançados de engenharia de software e ontologias para que possa ser realizada. O que a torna uma tarefa muito complexa e demorada para autores que não tenham as habilidades citadas.

Este trabalho de dissertação está situado na engenharia de aplicação da abordagem de construção de STIs proposta por (SILVA, 2011). Contribuindo com um modelo que automatize a EA de forma que seja possível construir ferramentas de autoria que tornem este processo menos custoso e complexo possibilitando que autores sem conhecimentos avançados de engenharia de software e ontologias possam construir STIs nesta abordagem.

4 SOLUÇÃO PROPOSTA

Este capítulo tem por objetivos apresentar um modelo para a Engenharia de Aplicação (EA) de uma abordagem de Linha de Produto de Software (LPS) semântica para construção de Sistemas Tutores Inteligentes (STIs) e a arquitetura de uma ferramenta que implementa este modelo.

4.1 INTRODUÇÃO

Com base no exposto nos Capítulos 1 e 3, propõe-se neste trabalho um modelo baseado em ontologia para automatizar a engenharia de aplicação de LPS para sistemas tutores inteligentes. De maneira específica pretende-se automatizar o processo de customização, instanciação e implantação de STIs de uma LPS, tornando transparente ao usuário a manipulação de ontologias e artefatos de software.

O modelo proposto utiliza ontologias para representar as funcionalidades e restrições de uma LPS genérica, as funcionalidades específicas para STIs, as decisões do autor em termos de quais funcionalidades farão parte do STI a ser gerado e as informações do aluno.

O processo de engenharia de aplicação da LPS é realizado por componentes que conduzem o autor pelas etapas de autenticação no sistema, seleção da LPS a ser instanciada, customização/configuração das funcionalidades do STI, validação, geração e implantação de um STI em um servidor Web.

Para validar o modelo proposto foi construída uma ferramenta que automatiza a geração de produtos em uma LPS. Tal ferramenta foi utilizada em um estudo de caso, apresentado no Capítulo 5, abrangendo a engenharia de aplicação de um STI a partir de uma LPS semântica.

4.2 FERRAMENTAS E TECNOLOGIAS UTILIZADAS

As principais ferramentas e tecnologias utilizadas pelo modelo proposto e sua implementação são descritas a seguir.

Protégé [1] ferramenta para manipulação de ontologias descritas em RDF(S) [2], OWL[3] e XML[4] *Schemas*. É utilizada diretamente na criação e composição das ontologias e regras de inferência associadas.

[1] <http://protege.stanford.edu>
[2] *Resource Description Famework* - <http://www.w3.org/RDF/>
[3] *OWL Web Ontology Language*-<http://www.w3.org/TR/owl-ref/>
[4] *Extensible Markup Language* - <http://www.w3.org/XML/>

Ontologia estrutura hierárquica utilizada para representar e armazenar conceitos de um determinado domínio, sendo a OWL a principal linguagem utilizada para definir ontologias na atualidade.

SWRL [5] é uma linguagem que combina OWL DL e OWL *Lite*, sub linguagens da linguagem de ontologia OWL, para descrever regras das ontologias, que são construídas em forma de implicação entre antecedente (corpo) e consequente (cabeça) (HORROCKS et al., 2004) .

Jena [6] é um *framework* de código-fonte livre desenvolvido pelo *HP Labs Semantic Web Research Group* [7] para a construção de aplicações semânticas. Este *framework* disponibiliza uma API [8] para manipulação de arquivos RDF, permite realizar leitura e escrita de ontologia descritas em RDF em diversos formatos (incluindo OWL), possui opções de armazenamento em banco de dados ou em memória, e um motor de execução de consultas SPARQL [9].

GKMT/JOINT [10] API para facilitar a manipulação de ontologias, seja com operações de inserção, remoção, recuperação e atualização de instâncias, como consultas SPARQL e raciocínio de regras semânticas (SWRL). Atualmente esta API faz parte da plataforma para o desenvolvimento de aplicações semânticas Joint.

Hibernate [11] é um *framework* de código-fonte livre de persistência em Java que realiza o mapeamento objeto-relacional entre os objetos da aplicação e um banco de dados relacional.

4.3 REQUISITOS

Uma ferramenta de autoria de STIs busca reduzir o esforço empregado no desenvolvimento dos STIs no que se refere ao tempo, custo, recursos utilizados e qualificações requeridas do autor/projetista para construir os STIs, permitindo que mais pessoas sejam capazes de participar do processo de projeto dos STIs (MURRAY, 2003).

Com base no exposto pode-se entender que os principais objetivos de uma ferramenta de autoria para STIs é tornar o processo de construção de STIs mais barato e fácil. Por sua vez os dois pontos citados estão intimamente ligados, pois as especificações do projeto das partes de um *software* são altamente dependentes das suposições feitas sobre quais usuários irão utilizá-lo.

---

5 *Semantic Web Rule Language*
6 http://jena.apache.org
7 <http://www.hpl.hp.com/research/about/semantic_web.html>
8 *Application Programming Interface*
9 <http://www.w3.org/TR/sparql11-overview/>
10 *Java Ontology INtegrated Toolkit* - http://nees.com.br/pt/joint/
11 <http://hibernate.org>

Os autores/projetistas de STIs, comumente, estão focados em quais funcionalidades seus STIs devem ter e não em seus aspectos arquiteturais e de implementação, logo uma das formas utilizadas para facilitar o processo de autoria dos STIs é construir uma ferramenta de autoria com uma interface gráfica com o usuário (GUI[12]) de tal forma que as ações necessárias para construção do STI sejam realizadas utilizando componentes gráficos simples de maneira que minimize a quantidade de conhecimento necessário para utilizá-la.

Pode-se citar como um elemento da GUI, que facilita a realização de uma tarefa, o *wizard*. Este elemento guia o usuário na realização de uma tarefa, por vezes complexa, através de um esquema passo-a-passo, bem definido, até que a tarefa seja concluída.

Em busca de aumentar a facilidade de uso de uma implementação do modelo proposto, o processo de engenharia de aplicação da LPS de STIs é estruturado em passos simples desde a escolha da linha de produtos a ser derivada até chegar à implantação do sistema. Isso é refletido no diagrama de casos de uso (Figura 11) onde há uma clara dependências entre os casos de uso.

O diagrama de casos de uso (Figura 11) possui quatro atores: *Author*, *Ontology Repository*, *Compiler* e *Web Server* descritos a seguir.

Author é o autor do STI a ser gerado. Mais especificamente é o usuário da ferramenta de autoria que possui o conhecimento e a autoridade necessário para escolher a linha de produtos de *software* (LPS) que deseja instanciar, customizar o modelo de *features* da LPS escolhida e requisitar a geração do STI.

Ontology Repository é o repositório das ontologias que representam os usuários da ferramenta, linhas de produtos e os STIs gerados pela ferramenta.

Compiler é quem provê funcionalidades à ferramenta de autoria para gerar o STI propriamente dito como, por exemplo, o arquivo *WAR*[13] do STI gerado para que possa ser implantado em um *web server*.

Web Server é o *software* e/ou *hardware* responsável por disponibilizar na *web* o STI gerado pela ferramenta de autoria para que os usuários finais possam utilizá-lo.

Ainda de acordo o diagrama da Figura 11, são apresentados os casos de uso a seguir.

[12] GUI - Graphical User Interface

[13] Web Application Archive é um arquivo compactado JAR utilizado para distribuir arquivos JSP (JavaServer Pages), Servlets do Java, classes Java, XML, HTML e outros recursos que fazem parte de uma aplicação Web.

Do Login o ator *Author* informa o nome e a senha de login para que o sistema faça a autenticação do usuário na ferramenta de autoria. Este caso de uso inclui a realização do caso de uso *Authenticate User*.

Authenticate User verifica junto à base de conhecimento da ferramenta (*Knowledge Base*) se o nome e senha informados pelo usuário são validos.

Select SPL o ator *Author* seleciona a linha de produto de *software* (LPS), de uma lista de LPS gerada pelo sistema, a partir da qual deseja gerar o seu STI e informa o nome do tutor a ser gerado. Este caso de uso tem como pré-requisito o caso de uso *Do Login* e inclui a realização do caso de uso *List SPLs*.

List SPLs realiza uma consulta ao *Knowledge Base* sobre as LPSs existentes e as disponibiliza ao autor.

Customize Product o ator *Author* seleciona as *features* que o produto deve ter (customização do produto) a partir do modelo de *features* da LPS selecionada. Este caso de uso tem como pré-requisito o caso de uso *Select SPL* e inclui a realização do caso de uso *Generate Feature Model*.

Generate Feature Model gera o modelo de *features* da linha de produtos selecionada pelo autor.

Request Product Validation requisita a verificação da ontologia do produto customizado. Este caso de uso tem como pré-requisito o caso de uso *Customize Product* e inclui a realização do caso de uso *Validate Product*.

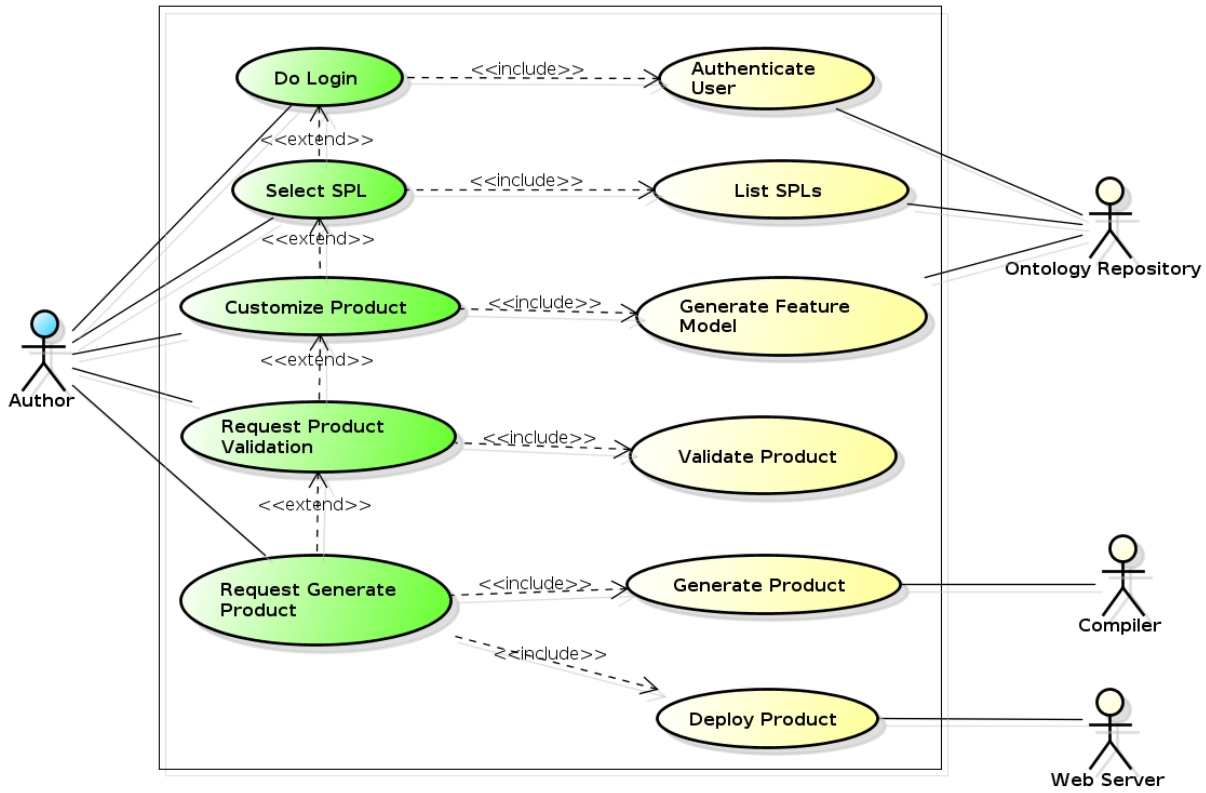
Validate Product faz uma verificação da ontologia do produto (neste trabalho é um STI) em busca de inconsistências e as informa à ferramenta para que, caso seja necessário, sejam corrigidas.

Request Generate Product requisita a geração do produto customizado e sem erros e a sua implantação no *web server*. Este caso de uso tem como pré-requisito o caso de uso *Request Product Validation* e inclui a realização dos casos de uso *Generate Product* e *Deploy Product*.

Generate Product busca os componentes do produto referentes ao modelo de *features* customizado e verificado e gera o produto propriamente dito pronto para ser implantado.

Deploy Product implanta o produto gerado no seu respectivo ambiente, como por exemplo, o *web server* utilizado.

Figura 11 – Diagrama de caso de uso do modelo proposto



Fonte: Autor desta dissertação, 2015.

4.4 MODELO PROPOSTO

A solução proposta é composta de um modelo de interação das ontologias ao qual representa semanticamente as LPSs no contexto de STIs, as decisões do autor/projetista, os STIs instanciados e as informações de aprendizes e do domínio do STI; e um modelo computacional para a EA de uma LPS semântica para construção de STIs. Esses modelos são apresentados nas sessões a seguir.

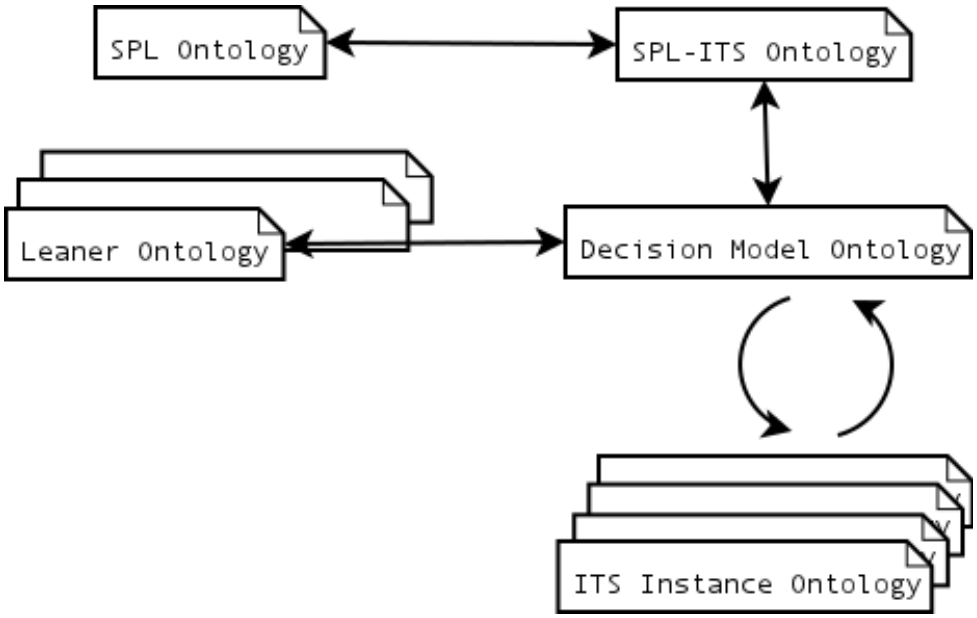
4.4.1 Modelo de Interação das Ontologias

As ontologias permitem descrever de forma semântica e consistente tanto o conhecimento dos STIs quanto os artefatos da LPS. Assim, o uso de ontologias provê uma forma geral de representar instâncias de famílias de STIs em diferentes domínios com diferentes requisitos.

Além disto, elas possibilitam separar o conhecimento da implementação do sistema permitindo que artefatos da LPS sejam independentes do domínio do STI.

Pelas razões citadas é importante utilizar ontologias que representem os artefatos da LPS, os requisitos de um STI e a especificação (i.e., decisões) do STI. Para tanto, foram combinadas ontologias em um modelo que permite descrever LPS no contexto de STIs (ver Figura 12).

Figura 12 – Modelo de interação das ontologias utilizadas neste trabalho



Fonte: Autor desta dissertação, 2015.

A ontologia *SPL* é utilizada como um meta-modelo para a especificação semântica do modelo de *features* utilizando o método de análise de domínio *Feature Oriented Domain Analysis* (FODA (KANG et al., 1990)). Com esse modelo é possível criar ontologias de qualquer domínio da aplicação utilizando as especificações FODA.

A ontologia *SPL-ITS* ou simplesmente *ITS* (*Intelligent Tutoring System*) é uma ontologia baseada na ontologia *SPL* (*Software Product Line*) onde seus indivíduos representam os componentes do modelo de *features* de uma família de STIs, permitindo que mudanças possam ser feitas independentemente dos domínios de conhecimento dos STIs.

A ontologia *Decision Model* representa o modelo decisões do autor acerca do produto a ser gerado. Mais especificamente, esta ontologia relaciona cada *feature* da LPS com um estado "selecionado"(i.e., indica que a *feature* será incluída no produto a ser instanciado) ou "eliminado"(i.e., indica que a *feature* não será incluída no produto a ser instanciado). No contexto dos STIs, esta ontologia modela as escolhas realizadas pelo autor sobre quais funcionalidades da família de STIs (representadas pela ontologia *ITS*) irão fazer parte do produto instanciado.

No contexto de STIs, há um foco no estudante e no ensino individualizado, por isso é utilizada a ontologia *Learner* (em português Aprendiz) para prover um mecanismo de representação dos estudantes e relacionar cada um deles a um conjunto de *features* e assim virtualmente cada estudante pode ter um STI personalizado. Também é possível utilizar em conjunto outras ontologias educacionais que representem o conhecimento do domínio do STI a ser instanciado.

Com estas quatro ontologias é possível configurar as características de cada produto de uma linha de STIs, sendo que para realizar a correta instanciação das ontologias dos produtos a serem gerados de forma a abstrair sua complexidade é necessária uma ferramenta que torne transparente ao usuário a manipulação das ontologias citadas e permita a validação de possíveis erros na configuração do produto.

Este modelo é flexível, de forma que se for necessário instanciar produtos de uma LPS em outro contexto diferente de STIs basta substituir as ontologias *SPL-ITS* por um outra que seja derivada de LPS e a ontologia *Learner* por outra(s) do domínio especifico da LPS que se quer instanciar.

O modelo computacional que permite construir uma ferramenta para automatizar a EA de uma abordagem de LPS que utiliza esse modelo de ontologias, é apresentado na seção a seguir.

4.4.2 Modelo Computacional

O modelo proposto tem foco na engenharia da aplicação, principalmente na correta instanciação da ontologia que descreve as funcionalidades do produto a ser gerado.

De modo geral, esse modelo baseia-se em um estilo arquitetural heterogêneo que combina os estilos arquiteturais *pipes and filters*, onde os dados de saída de um componente da arquitetura alimentam a entrada do próximo componente até que o processamento do dado seja concluído; componentes independentes, onde a arquitetura é composta por componente lógicos funcionais, que não possuem dependência direta entre si e possuem interfaces de comunicação bem definidas; em camadas, onde as responsabilidades do sistema são bem definidas em camadas.

Como apresentado na Figura 13, o estilo arquitetural utilizado como arquitetura de referência foi o estilo centrado em conhecimento que baseia-se no estilo centrado em dados, mais especificamente o estilo *repository* que tem como principal objetivo separar a lógica da aplicação do acesso aos dados e ao mesmo tempo permitir o compartilhamento dos dados entre os componentes, porém como seus dados possuem uma natureza essencialmente semântica, no contexto deste trabalho, foi definido este estilo arquitetural como centrado em conhecimento.

A representação lógica de mais alto nível do modelo, apresentada na Figura 13, é

composta de três componentes independentes que acessam uma mesma base de conhecimento.

Knowledge Base é a base de conhecimento, responsável por manipular os SGBDs[14] relacionais e principalmente baseados em ontologia; também tem a função de disponibilizar uma interface de alto nível aos outros componentes do modelo. Esse componente representa a principal fonte de integração entre os demais componentes do modelo, através do compartilhamento de dados. Sua arquitetura interna é detalhada na Seção 4.4.2.4.

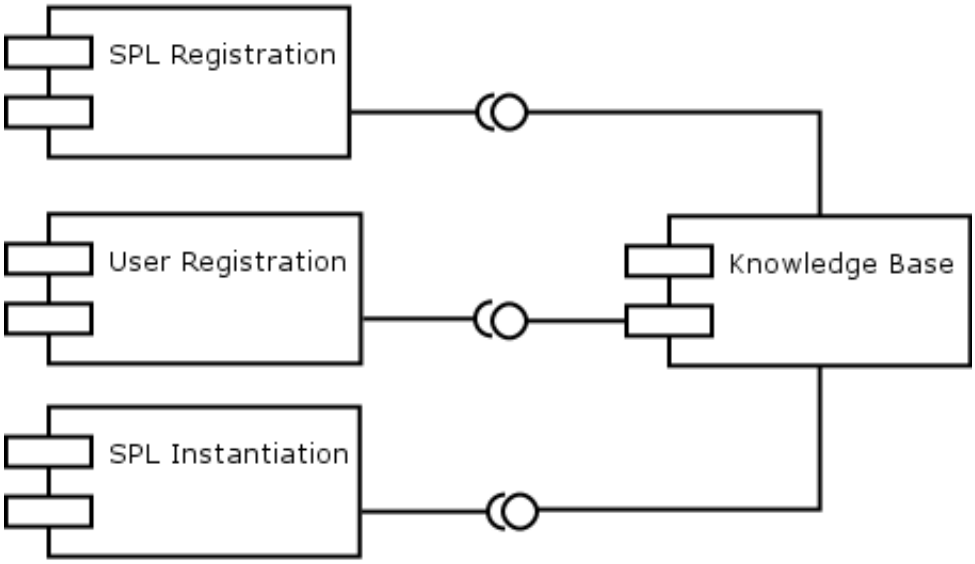
SPL Registration é o componente de cadastro de LPSs, tem a responsabilidade de permitir a inserção de novas LPSs na base de conhecimento (i.e., *Knowledge Base*). Após o cadastro, as LPSs ficam disponíveis aos autores/projetistas para instanciar novos produtos.

User Registration é o componente de cadastro de usuários (i.e., autores/projetistas), tem a responsabilidade de permitir a inserção de novos usuários na base de conhecimento (i.e., *Knowledge Base*). Os usuários cadastrados podem instanciar novos produtos a partir de uma LPS já cadastrada ou cadastrar uma nova LPS.

SPL Instantiation é o componente de instanciação de produtos da LPS, responsável por automatizar as tarefas da engenharia de aplicação e conduzir o autor/projetista pelas decisões desse processo, mais especificamente permite que um autor selecione uma das LPSs cadastradas, customize (i.e., configure, personalize) um produto da LPS selecionada, valide a configuração do produto, gere e implante o produto instanciado em um servidor Web. Sua arquitetura interna é detalhada na Seção 4.4.2.1.

[14] Sistema gerenciador de banco de dados

Figura 13 – Representação lógica de mais alto nível do modelo computacional



Fonte: Autor desta dissertação, 2015.

Tendo em vista um caráter incremental de desenvolvimento, o estilo arquitetural centrado em conhecimento foi utilizado para facilitar a integração entre os componentes. Essa facilidade de integração vai além dos componentes previstos inicialmente, possibilitando adicionar futuramente outros componentes complementares, e.g., componente gerador de artefatos da LPS, componente de recomendação de *features*.

Nesse modelo, cada componente implementa parte das funcionalidades e a comunicação entre eles ocorre através do compartilhamento de uma mesma base de conhecimento.

Nas próximas seções, são detalhados os componentes de instanciação da LPS (i.e, *SPL Instantiation*) e a base de conhecimento (i.e., *Knowledge Base*) por estarem diretamente relacionados a solução do problema proposto nesta dissertação.

4.4.2.1 Detalhamento do Componente *SPL Instantiation*

SPL Instantiation permite realizar a derivação das LPSs cadastradas, ou seja, instanciar produtos a partir de uma LPS de acordo com a configuração feita pelo autor/projetista. Ele possui componentes internos responsáveis pela autenticação do usuário, seleção da LPS a ser instanciada, customização (i.e., configuração) de um produto da LPS selecionada, validação da configuração do produto, geração e implantação do produto instanciado em um servidor Web. A

Figura 14 apresenta a arquitetura do *SPL Instantiation*, composta pelos seguintes componentes arquiteturais:

Login é o componente responsável por realizar a autenticação do usuário no sistema, utiliza a base de conhecimento (i.e., *Knowledge Base*) para verificar se o nome do usuário e a senha estão corretos. Após corretamente autenticado no sistema, o usuário é direcionado para o componente de seleção da LPS (i.e., *SPL Selection*).

SPL Selection é o componente responsável disponibilizar ao usuário as LPSs cadastradas na base de conhecimento (i.e., *Knowledge Base*), disponibilizá-las ao usuário para que possa escolher a partir de qual LPS será instanciado o produto e definir o nome do produto. Após a definição de qual o nome do produto e qual LPS será utilizada, são passadas ao componente *Product Customization* as URIs (i.e., identificadores) [15] das ontologias da LPS e do modelo de decisão do produto (i.e., *Product Decision Model*).

Product Customization é o componente responsável por disponibilizar ao usuário o modelo de *features* da LPS selecionada permitindo incluir ou remover *features* no produto a ser instanciado. A partir desse componente o usuário pode também solicitar a validação do modelo de *features* e, caso após a validação do produto não tenha erros, pode solicitar a geração do produto customizado.

Feature Validation é responsável por fazer a validação do modelo de *features* do produto, ou seja, verifica as dependências, restrições e outros requisitos das *features* do produto que está sendo customizado. Para verificar o modelo de *features* do produto, utiliza um motor de inferência (*engine*) para regras SWRLs sobre o arquivo OWL gerado, isto confere maior flexibilidade para alterar as regras de validação. Caso existam erros, retorna ao *Product Customization* exibindo mensagens com os erros encontrados para que sejam corrigidos pelo usuário. Caso contrário retorna um sinal que o produto está válido.

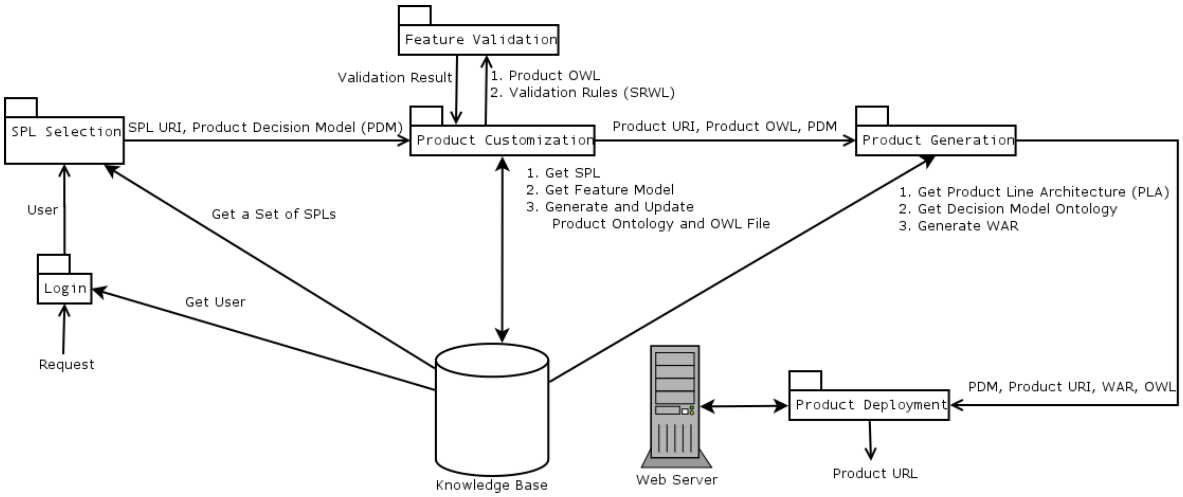
Product Generation é responsável por gerar o produto a partir da suas ontologias, instanciar os componentes referentes as *features* selecionadas e gerar ao final um produto (e.g., para Java Web um arquivo WAR) pronto para ser implantado no servidor Web. O WAR, OWL, URI e PDM do produto são passados para o componente *Product Deployment*.

[15] *Uniform Resource Identifier* - <http://www.w3.org/Addressing/>

Product Deployment é responsável por implantar o produto (consiste dos arquivos WAR e OWL) no servidor Web e disponibilizar a URL[16] do produto implantado ao usuário. Essa é a etapa final da engenharia de aplicação da LPS pois disponibiliza ao usuário final o produto pronto para uso, ou seja, instanciado e implantado junto com a sua descrição semântica.

Web Server é o servidor Web onde são implantados os produtos, responsável por disponibilizá-los na internet para os usuários.

Figura 14 – Detalhamento do *SPL Instantiation*



Fonte: Autor desta dissertação, 2015.

4.4.2.2 Detalhamento do Componente *Product Customization*

O componente *Product Customization* utiliza a base de conhecimento (i.e., *Knowledge Base*) para disponibilizar o modelo de *features* da LPS na interface gráfica com o usuário (*GUI*), gerar um arquivo OWL com as ontologias do produto a ser instanciado e opcionalmente realizar uma validação simples das *features* que estão sendo selecionadas pelo usuário (i.e., validação axiomática do produto).

A arquitetura do *Product Customization* (Figura 15) é composta pelos componentes *Feature Selection and Axiomatic Validation* e *Product OWL Generation*, ambos compartilham a mesma base de conhecimento (i.e., *Knowledge Base*). Esses componentes, são descritos a seguir:

[16] *Universal Resource Location*

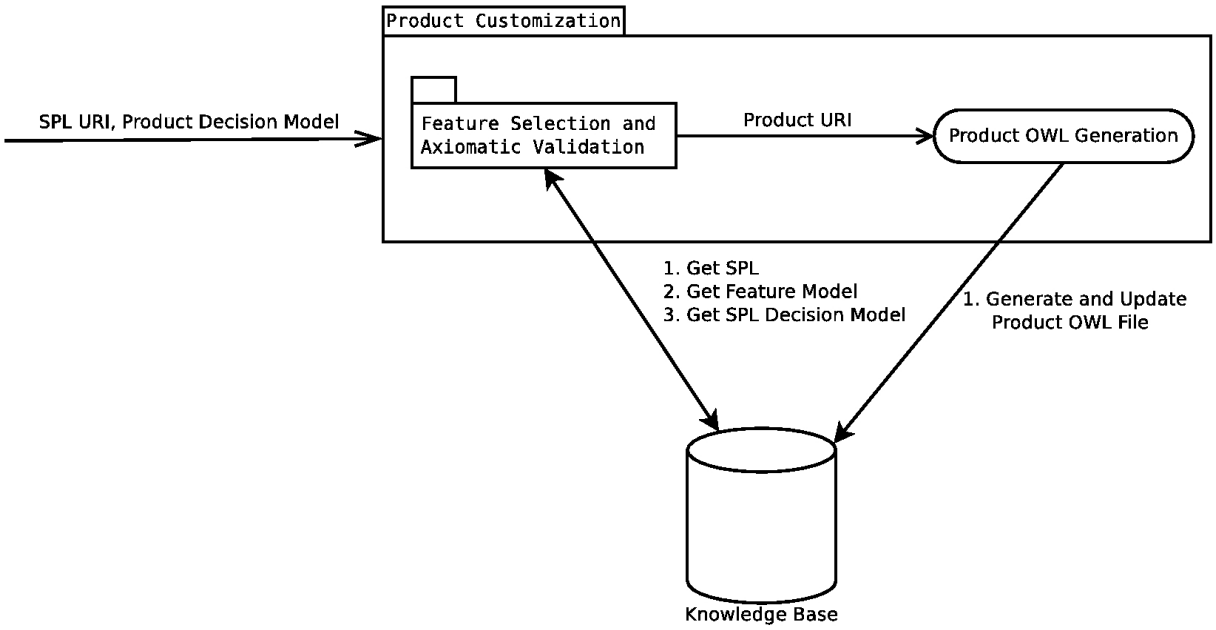
Feature Selection and Axiomatic Validation é responsável por alterar o *Product Decion Model* de acordo com as *features* selecionadas pelo usuário e opcionalmente realizar a validação axiomática (regras básicas do modelo de *features*) do produto, informando as inconsistências para que sejam solucionadas pelo usuário (ver Quadro 2).

O componente *Axiomatic Validation* provê uma validação das restrições do modelo de *features* ao mesmo tempo em que as mesmas estão sendo selecionadas, mas é um módulo opcional pois suas verificações podem impactar negativamente no desempenho da aplicação devido ao processamento das regras de validação, podendo suas funções serem desempenhadas pelo componente *Feature Validation* após o término da configuração do modelo de *features*.

AxiomaticValidation tem um comportamento semelhante ao do padrão de projetos *Observer*, isto é, à medida que ocorre a seleção de *features* por parte do componente *Feature Selection* (sujeito observado), o componente *Axiomatic Validation* é notificado para verificar a integridade do conjunto selecionado, de acordo com as regras do modelo de *features* da LPS.

Product OWL Generation é responsável por gerar a ontologia OWL do produto, atualizar o arquivo OWL existente e persistir a ontologia do produto no banco utilizando a *Knowledge Base*. Ao concluir a geração do produto envia a URI do OWL para o componente *Feature Validation* para que possam ser executadas as regras de validação do modelo de *features* do produto.

Figura 15 – Detalhamento do *Product Customization*



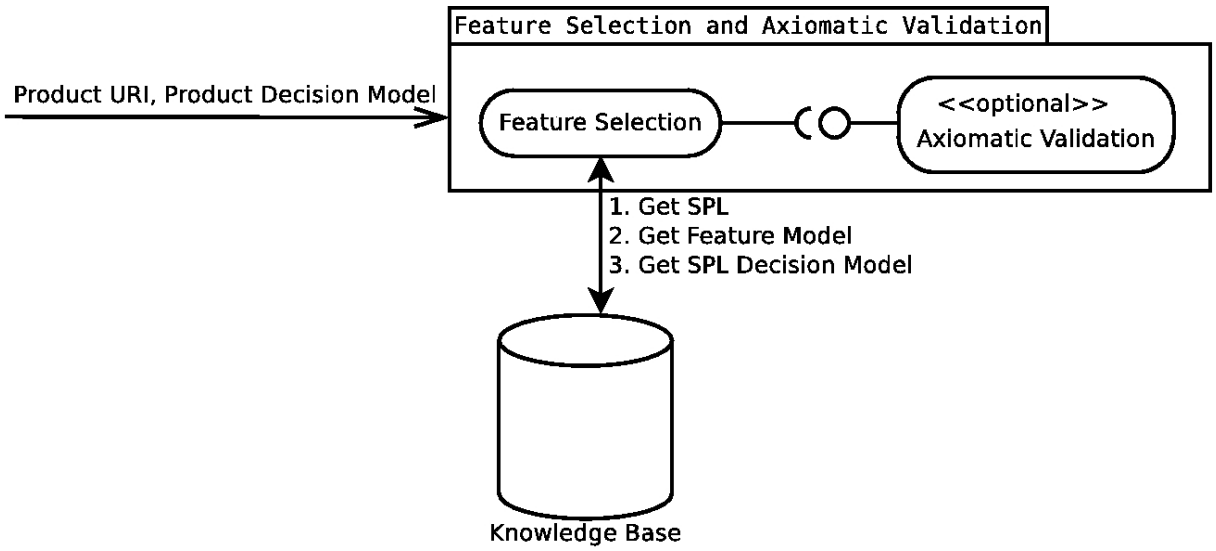
Fonte: Autor desta dissertação, 2015.

No diagrama apresentado na Figura 16, é apresentada a arquitetura do *Feature Selection and Axiomatic Validation*. Sua arquitetura é composta pelos componentes: *Feature Selection e Axiomatic Validation* descritos abaixo:

Feature Selection é responsável por acessar a base de conhecimento (i.e., *Knowledge Base*), obter a LPS de acordo com o URI passada, obter suas *features* e alterar o *Product Decion Model* de acordo com as *features* selecionadas pelo usuário.

Axiomatic Validation é responsável por validar axiomaticamente o *Product Decion Model* e caso existam erros informa ao usuário para que sejam corrigidos. Caso não existam erros axiomáticos é chamado o componente *Product OWL Generation* para gerar ou atualizar o arquivo OWL do produto de acordo com a ontologia do produto persistida na *Knowledge Base*.

Figura 16 – Detalhamento do *Feature Selection and Axiomatic Validation*



Fonte: Autor desta dissertação, 2015.

Quadro 2 – Algoritmo de verificação de erros pelo *Axiomatic Validation*

Entrada: *feature* selecionada, Modelo de *features*, Modelo de decisão do produto (*Product Decion Model*)
Saída: Mensagem de erro na seleção de uma *feature* do modelo
início
  se *FeatureSelection informou que uma feature foi selecionada* **então**
    AxiomaticValidation verifica se a *feature* esta de acordo com as restrições básicas do modelo de *features*;
    se *Existe erro* **então**
      retorna *mensagem de erro*
    fim
  senão
    aguarda *FeatureSelection* informar que uma *feature* foi selecionada;
  fim
fim

Fonte: Autor desta dissertação, 2015.

4.4.2.3 Detalhamento do Componente *Feature Validation*

Durante o processo de customização do produto pode haver inconsistências no modelo de *features*, tais como: *features* de seleção obrigatória (*madatory*) não selecionadas, duas *features* alternativas (*alternative*) entre si selecionadas, *features* "pai" selecionadas com suas *features* "filhas" obrigatórias desmarcadas e vice-versa entre outras inconsistências que podem levar a

geração, implantação e disponibilização ao usuário de um STI com comportamento inconsistente, faltando algumas funcionalidades e/ou simplesmente inutilizável.

O cenário anteriormente citado não é interessante, portanto a validação das ontologias dos produtos a serem gerados é uma das funcionalidades que mais se destaca no processo de autoria de um STI, pois auxilia o autor/projetista na customização de seu STI através da detecção de erros na configuração do seu modelo de *features*. Com isso, é possível impedir que um autor possa gerar e implantar um produto inconsistente no servidor. Outro aspecto importante do processo de validação das ontologias é verificar se o tutor gerado está em conformidade com o modelo de *features* de sua linha de produtos atualizada e sugerir melhorias a um STI existente.

O componente *Feature Validation* realiza a validação das ontologias através de um motor de inferência que executa regras semânticas (e.g., SWRL). Esse componente têm dois módulos *Product Validator* e *Rule Engine*, descritos a seguir:

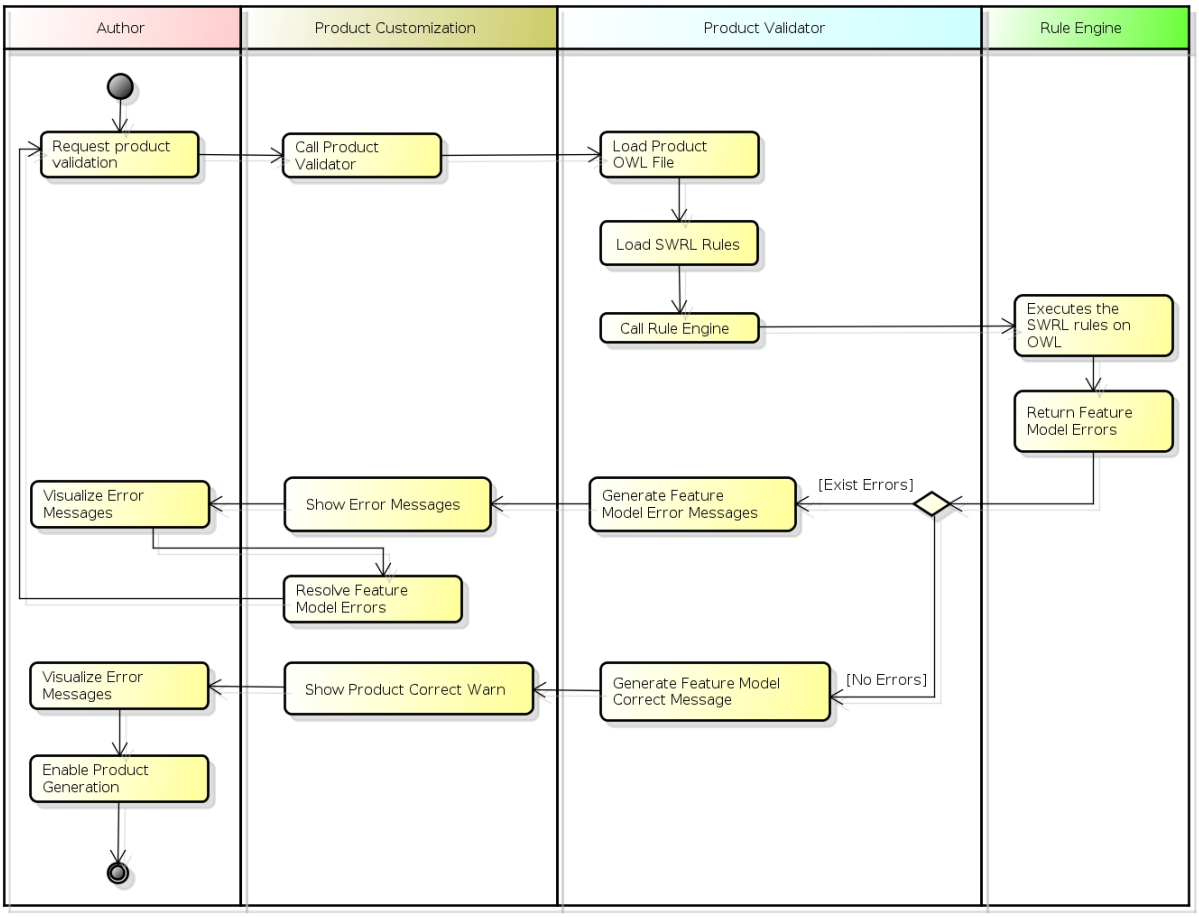
Product Validator responsável por invocar o motor de regras (*Rule Engine*) passando como parâmetro as ontologias OWL do produto a ser gerado e as regras SWRL a serem executadas, caso sejam detectados erros, *Product Validator* gera mensagens de erro a serem exibidas ao usuário.

Rule Engine responsável por executar as regras semânticas e retornar as *features* que foram "capturadas" pelas regras.

O processo de validação das ontologias do produto a ser gerado (apresentado na Figura 17) tem início quando o autor requisita a validação do modelo de *features*, na customização do produto. Essa ação chama o componente *Product Validator*, que carrega o arquivo OWL das ontologias do produto, as regras semânticas SWRL dos possíveis erros do modelo de *features* e faz um chamado ao componente *Rule Engine*.

O componente *Rule Engine* executa as regras semânticas e retorna ao *Product Validator* os erros e as *features* capturadas pelas regras, para que o componente de validação do produto (*Product Validator*) possa, caso existam erros, gerar as mensagens de erro ou caso contrário gerar um aviso que o produto está correto e repassa-las ao *Product Customization* para que exiba estas mensagens a fim de que o autor possa visualiza-las e fazer as possíveis correções ou, no caso do produto estar correto, passar para a geração do produto (*Product Generation*).

Figura 17 – Diagrama de atividades do processo de validação do produto (*Feature Validation*)



Fonte: Autor desta dissertação, 2015.

4.4.2.4 Detalhamento do Componente *Knowledge Base*

O componente *Knowledge Base* representa a principal fonte de integração entre os demais componentes do modelo, essa integração ocorre através do compartilhamento de dados desta base de conhecimento. Sua arquitetura é apresentada na Figura 18 e baseia-se em três camadas: *Controller*, *Model and Persistence* e *Database* descritas a seguir.

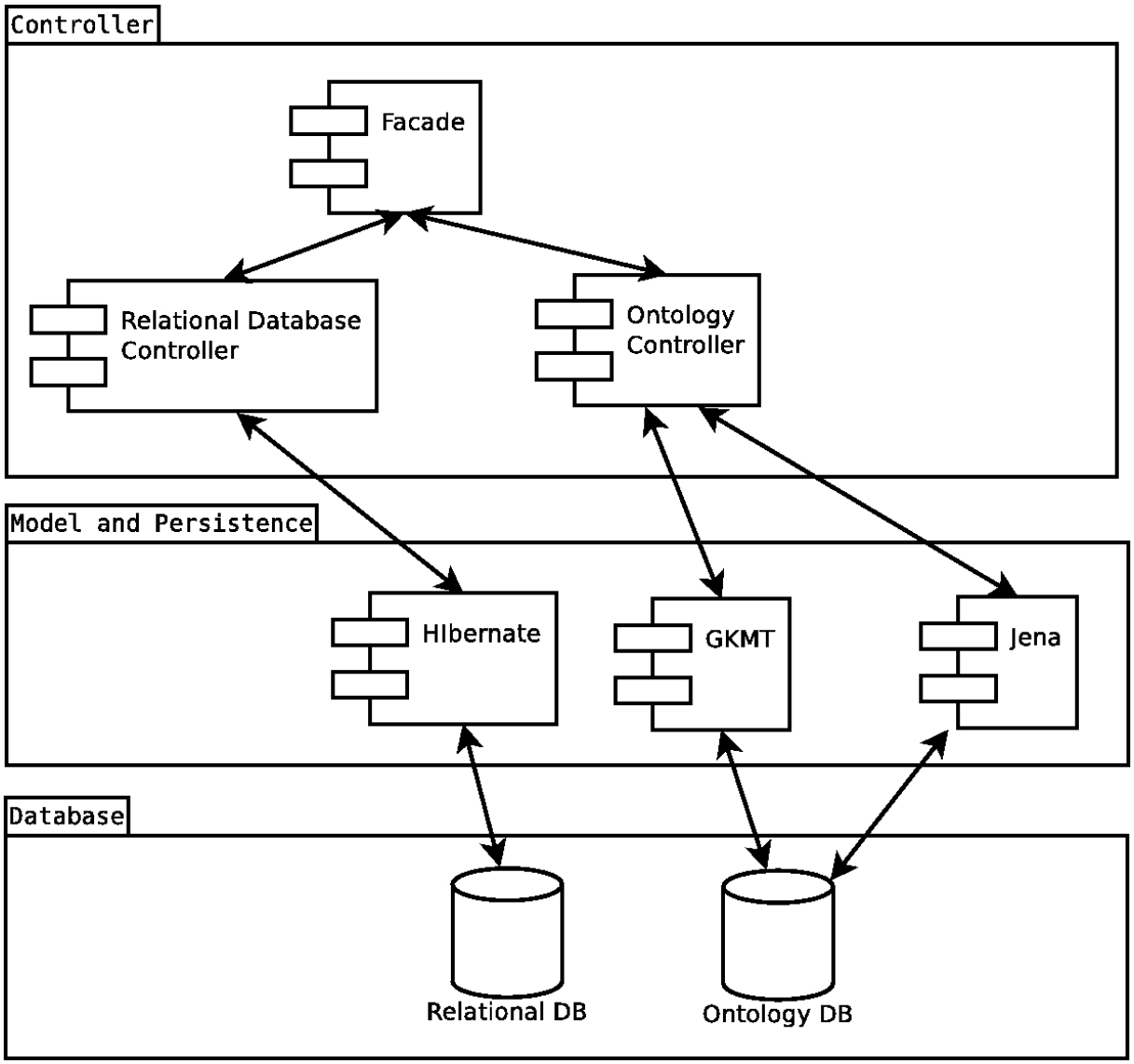
Controller camada responsável por gerenciar e repassar as requisições de manipulação da base de conhecimento. É composta pelos componentes: *Facade*, responsável por disponibilizar uma interface única para a manipulação da base de conhecimento pelos componentes externos; *Relational Database Controller*, responsável por gerenciar e repassar as requisições de manipulação dos registros do banco de dados relacional à camada *Model and*

Persistence; *Ontology Controller*, responsável por gerenciar e repassar as requisições de manipulação das ontologias à camada *Model and Persistence*.

Model and Persistence camada responsável pela manipulação dos bancos de dados existentes na camada *Database* provendo um maior nível de abstração e flexibilidade em relação aos SGBDs utilizados. Essa camada é composta pelos *framewoks* de persistência *Hibernate*, realiza o mapeamento objeto-relacional entre os objetos da aplicação e o banco de dados relacional (*Relational DB*); *GKMT*, permite manipular as ontologias e repositórios do *Ontology DB* de forma orientada à objetos; *Jena*, provê um uma API para OWL permitindo a geração e manipulação de arquivos OWL pelo *Ontology Controller*.

Database camada responsável por disponibilizar os SGBDs, sem qualquer lógica de negócio. é composta pelo *Relational DB*, é um banco de dados relacional utilizado apenas para persistência de dados relacionados aos usuários do sistema e suas permissões; *Ontology DB*, é um banco de dados para ontologias.

Figura 18 – Detalhamento da base de conhecimento (*Knowledge Base*)



Fonte: Autor desta dissertação, 2015.

4.4.2.5 Processo de Instanciação de um Produto

O processo geral para instanciar produtos de uma LPS semântica utilizando o modelo proposto é apresentado nos diagramas de sequência apresentados na Figura 19 e na Figura 20. Nesse processo propõe-se o uso de tecnologias semânticas para automatizar o processo de engenharia de aplicação e obter um conjunto de especializações do modelo de *features* da LPS escolhida para atender a um determinado conjunto de requisitos do autor/projetista.

Inicialmente o autor/projetista autentica-se através do componente *Login* e é redirecionado para o componente de seleção de LPS (*SPL Selection*) onde escolhe qual a LPS a ser

derivada e define o nome do produto a ser gerado. Então, o autor/projetista é redirecionado para o componente *Product Customization* para realizar a customização do produto a ser gerado

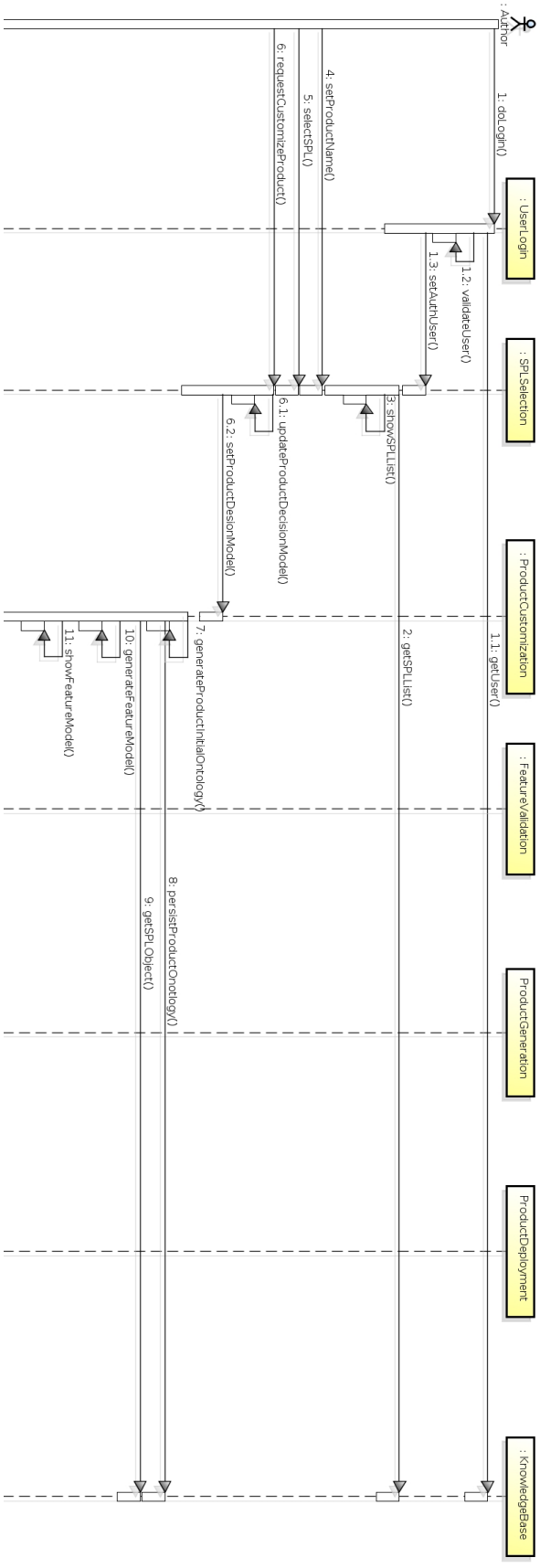
No processo de customização, um subconjunto de funcionalidades é selecionado com base em restrições especificadas de acordo com as capacidades de cada sistema. O modelo de decisões gerado por esse processo, descrito por ontologias OWL, deve ser consistente, ou seja, refletir todas as características bem como as restrições que excluem e incluem funcionalidades nos modelos de *features* anotados semanticamente. Esse modelo de *features* derivado é uma especialização do modelo de *features* da LPS escolhida no *SPL Selection*.

De forma geral, cada produto gerado é um subconjunto do modelo de *features* da LPS original. Em termos de raciocínio baseado em ontologias, o objetivo é determinar se as instâncias de uma ontologia são consistentes com relação às propriedades de anotação definidas no modelo de *features*. Esse processo de validação da ontologia é realizado pelo componente *Feature Validation*.

Se ao final do processo de validação do produto o resultado for uma ontologia inconsistente, ou seja, que não cumpre com as restrições do modelo de *features* da LPS, é necessário voltar à customização do produto (*Product Customization*) e refinar o modelo a fim de satisfazer as inconsistências descobertas.

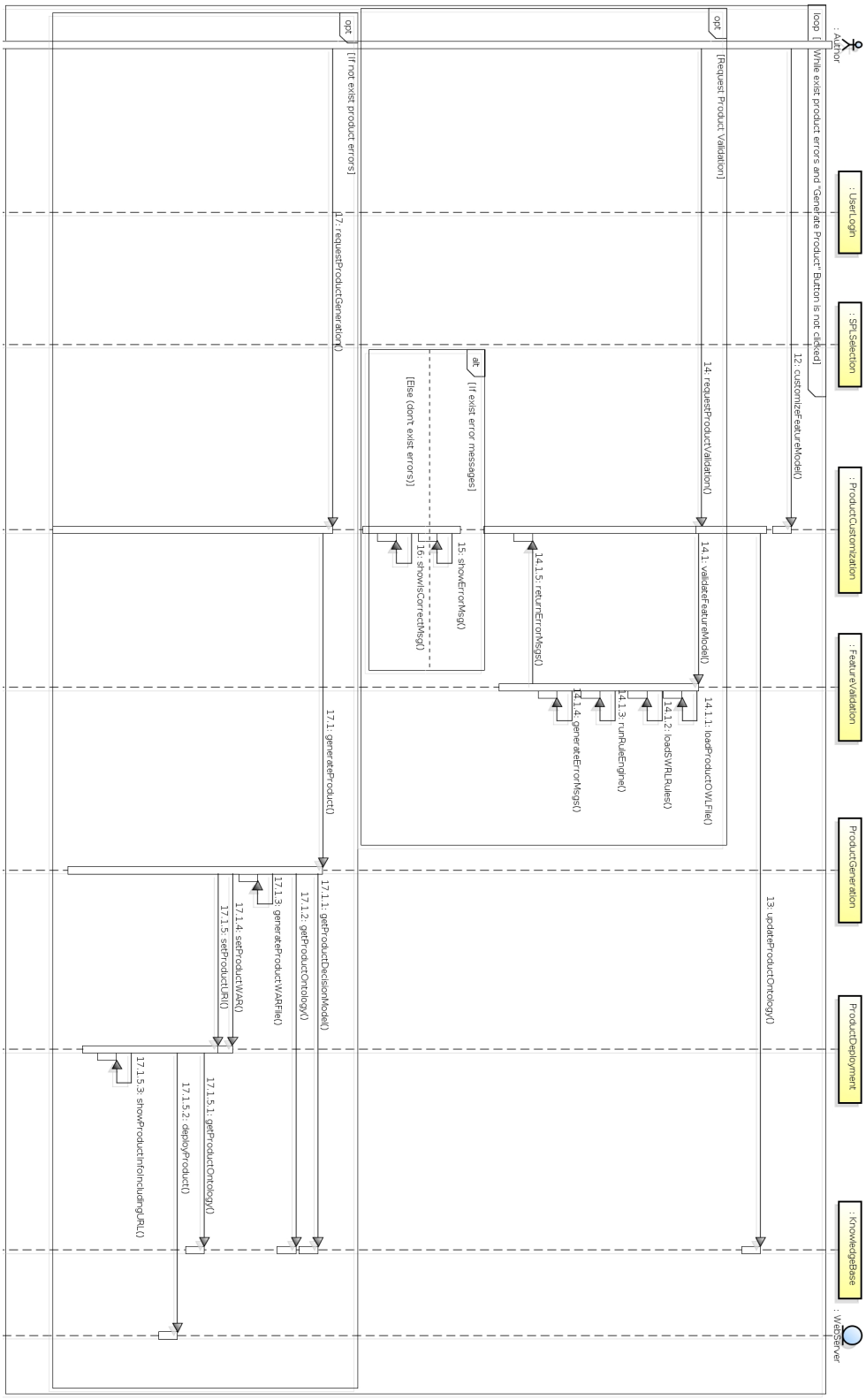
Caso contrário, se o resultado da validação do produto for uma ontologia consistente, pode-se dar início ao processo de descoberta e integração dos componentes de software do produto (realizado pelo *Product Generation*) e sua implantação no servidor web (realizado pelo *Product Deployment*). Após a implantação do produto, é disponibilizado ao autor/projetista a URL (i.e., endereço de acesso) do produto implantado

Figura 19 – Processo de Instanciação de um produto pelo *Product Instantiation* (Parte I)



Fonte: Autor desta dissertação, 2015.

Figura 20 – Processo de Instanciação de um produto pelo *Product Instantiation* (Parte II)



Fonte: Autor desta dissertação, 2015.

4.5 ASPECTOS DE IMPLEMENTAÇÃO

A implementação do modelo proposto neste trabalho foi realizada utilizando a linguagem de programação Java[17], o framework MVC JSF 2[18] e a biblioteca de componentes visuais para JSF *PrimeFaces*[19] no ambiente de desenvolvimento *NetBeans*[20]. Esta implementação tem três módulos: *User Registration*, *SPL Registration* e *SPL Instantiation*.

O componente *User Registration* foi implementado como um CRUD[21] de usuário simples, ou seja, permite as operações de criação, recuperação, atualização e exclusão de usuários do sistema utilizando um banco de dados relacional MySQL.

O componente *SPL Registration* permite inserir novas ontologias (com formato OWL) de uma LPS especifica (e.g., LPS de STIs da UFAL) no sistema e realizar as operações de recuperar, atualizar e deletar ontologias de LPSs já cadastradas. Associado as ontologias cadastradas, também é cadastrado um arquivo de formato WAR com os artefatos de software de sua respectiva LPS.

Ambos os componentes *User Registration* e *SPL Registration* utilizam o componente *KnowledgeBase* sendo que *User Registration* persiste os usuários do sistema em um banco relacional MySQL e *SPL Registration* persiste as ontologias em um repositório semântico Sesame.

4.5.1 Aspectos de Implementação do Componente *SPL Instantiation*

A arquitetura do componente *SPL Instantiation*, apesentada na Figura 21, assemelha-se a arquitetura do modelo proposto na Figura 14 implementando suas respectivas funcionalidades.

Os componentes *Login*, *SplSeletion* e *ProductCustomization* foram implementados com uma arquitetura em camadas onde na camada de controle é composta por classes *managed beans*, responsáveis por intermediar a comunicação em as páginas JSF e as classes de modelo, além de captar eventos e processá-los de forma a realizar a correta chamada dos métodos das classes responsáveis pela lógica da aplicação.

Os componentes *FeatureValidation*, *ProductGeneration* e *ProductDeployer*, não interagem diretamente com a camada de visão da implementação, assim possuem apenas classes de

[17] http://java.sun.com

[18] JavaServer Faces - http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[19] www.primefaces.org

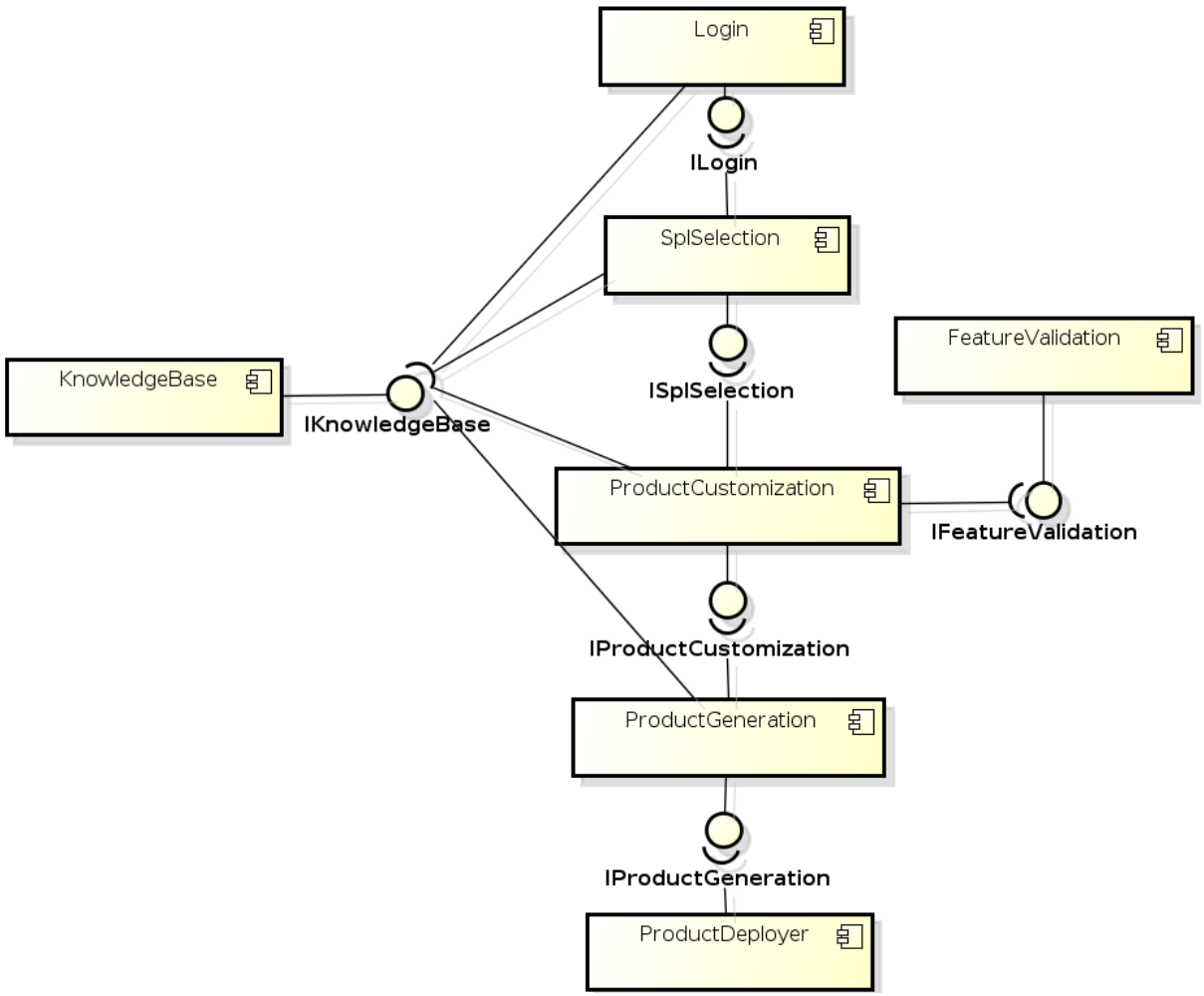
[20] www.netbeans.org

[21] acrónimo de *Create*, *Read*, *Update* e *Delete* na língua Inglesa, para as quatro operações básicas: criação, consulta, atualização e exclusão de dados do banco

modelo e lógica da aplicação além de suas interfaces de utilização para se comunicar com os outros componentes.

A maioria dos componentes do *SPL Instantiation* necessitam interagir com o componente *KnowledgeBase* para realizar suas operações, visto que os dados estão centrados na base de conhecimento em forma de ontologias que apenas podem ser alteradas através da interface provida pelo *KnowledgeBase*. Além da base de conhecimento os componentes de *SPL Instantiation* possuem classes de modelo em comum que representam, de forma orientada a objetos, os dados das ontologias.

Figura 21 – Diagrama de componentes do *SPL Instantiation*



Fonte: Autor desta dissertação, 2015.

4.5.2 Aspectos de Implementação do Componente *Feature Validation*

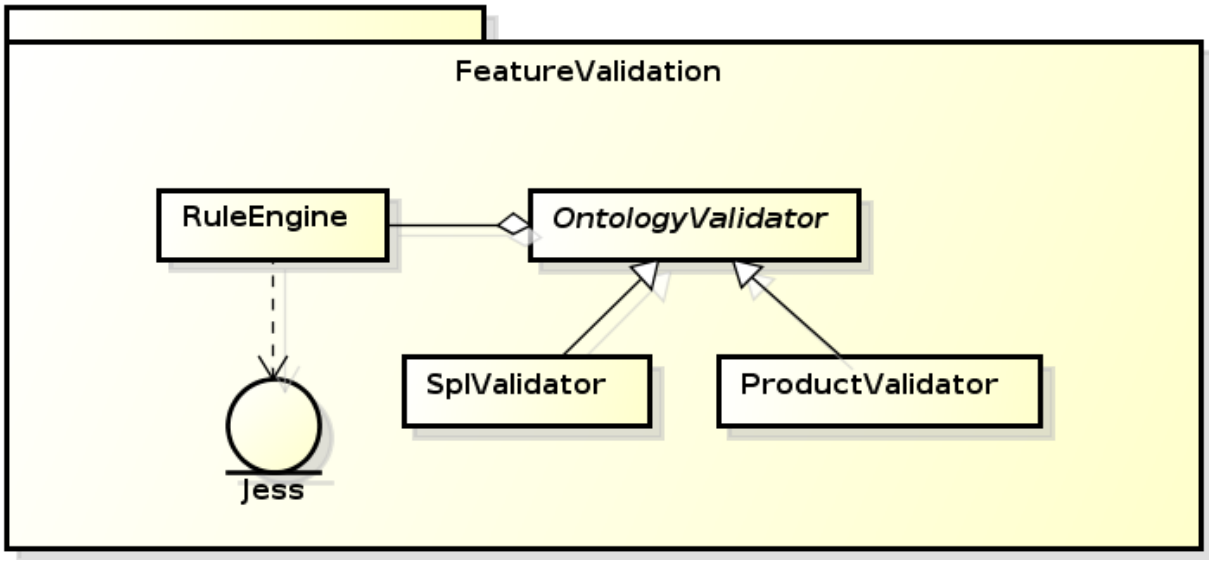
A arquitetura do componente *Feature Validation* apresentada na Figura 22, é composta pelos módulos *Rule Engine* e *Product Validator*, sendo o primeiro responsável por executar o motor de inferência das regras SWRL e o segundo por chamar o motor de regras (*Rule Engine*) passando os parâmetros corretos para detecção de erros na configuração de um produto da LPS e por gerar as mensagens de erro.

Os componentes arquiteturais descritos foram implementados em classes, onde a classe *ProductValidator* é uma classe filha da classe abstrata *OntologyValidator* que, apesar de não poder ser instanciada, tem métodos que realizam a verificação de um arquivo OWL a partir de um conjunto de regras SWRL com o auxilio de *RuleEngine* que faz uso da API do *Jess*.

A classe *OntologyValidator* permite ampliar as funcionalidades do componente de validação como, por exemplo, ao criar a classe filha *SplValidator* pode-se também verificar a estrutura da linha de produtos em busca de erros e possíveis otimizações além de gerar as mensagens adequadas.

A classe concreta *ProductValidator* é a única acessada diretamente pelo componente *SPL Instantiation*. Ela possui, além dos métodos herdados de *OntologyValidator*, métodos que geram as mensagens corretas para cada erro encontrado na verificação dos produtos.

Figura 22 – Diagrama de classes do componente *Feature Validation*



Fonte: Autor desta dissertação, 2015.

Quando o componente *Feature Validation* é chamado para realizar a verificação do

modelo de *features* do produto a ser gerado, é solicitado ao componente *Knowledge Base* a criação de um arquivo OWL contendo a ontologia que representa o modelo de decisão do produto. Faz-se necessário criar o arquivo OWL por uma limitação do *Jess*, que não executa as regras SWRL diretamente sobre o repositório *Sesame*. Em seguida, inicia-se o processo de verificação do modelo de *features* do produto.

O processo de verificação da ontologia (OWL) do produto, é implementado pelo componente *Feature Validation* de acordo com o algoritmo apresentado no Quadro 3.

Como pode ser observado no Quadro 3, inicialmente é chamada a classe *ProductValidator*, passando como parâmetros o caminho do diretório da ontologia do produto (OWL) e das regras SWRL. Em seguida, *ProductValidator* carrega o arquivo OWL da ontologia do produto e as regras SWRL para a verificação do produto. Para cada regra SWRL carregada é chamada à classe *RuleEngine*, passando o arquivo OWL e as regra SWRL como parâmetros. Então, sobre o OWL são aplicadas as regras SWRL para que sejam capturadas as *features* que atendem às regras. A *RuleEngine* retorna o erro e as *features* capturadas, e retorna ao *ProductValidator* para que possa armazenar a descrição do erro e as *features* capturadas. Para cada erro armazenado, *ProductValidator* gera uma mensagem e a armazena. Caso existam mensagens de erro armazenadas, estas são retornadas. Caso contrário, retorna a mensagem de que o produto está correto (i.e., válido).

Quadro 3 – Algoritmo de verificação de erros pelo *Feature Validation*

Entrada: Caminho do diretório da ontologia do produto (OWL) e o Caminho do diretório das regras de verificação (SWRL)
Saída: Mensagem do resultado da validação do produto
início
    ProductValidator carrega o arquivo OWL da ontologia do produto;
    ProductValidator carrega as regras SWRL para verificação do produto;
    para *cada regra SWRL carregada* **faça**
        RuleEngine executa a regra sobre o arquivo OWL;
        RuleEngine captura as *features* que atendem as regras;
        RuleEngine retorna o erro e as *features* capturadas ;
        ProductValidator armazena a descrição do erro e as suas respectivas *features* capturadas ;
        para *cada erro armazenado* **faça**
            ProductValidator gera a mensagem de erro;
            ProductValidator armazena a mensagem de erro;
        fim
    fim
    se *Existe mensagem de erro armazenada* **então**
        retorna *Retorna as mensagens de erro armazenadas*
    senão
        retorna *Retorna a mensagem de produto correto*
    fim
fim

Fonte: Autor desta dissertação, 2015.

A classe *Product Validator* utiliza um conjunto de regras SWRL para realizar a detecção de erros na configuração do modelo de *features* da LPS (descrito pela ontologia *DecisionModel*). Essas regras são executadas pelo motor de inferência da classe *RuleEngine* sobre as ontologias do produto a ser instanciado.

Os prefixos SPL e decisionModel fazem, respectivamente, referência às ontologias que modelam a LPS e as decisões do autor tais como quais *features* irão fazer parte ou não do produto final de uma determinado produto da LPS.

A regra semântica SWRL apresentada no Quadro 4, captura a *feature* que deve obrigatoriamente fazer parte do produto (tipo *MANDATORY*) e não está selecionada (propriedade *CurrentState ELIMINATED*).

Quadro 4 – Regra SWRL de validação do produto 1

```
1 SPL:Feature(?f) ^ SPL:hasRelationType(?f, "mandatory") ^
2 decisionModel:CurrentState(?f, "ELIMINATED")-> ↩
     sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 5, captura a *feature* selecionada (*CurrentState SELECTED*) onde a sua *feature* pai não está selecionada.

Quadro 5 – Regra SWRL de validação do produto 2

```
1 SPL:Feature(?p) ^ decisionModel:CurrentState(?p, "↩
     ELIMINATED") ^
2 SPL:isParentOf(?p, ?f) ^ decisionModel:CurrentState(?f, "↩
     SELECTED")
3 -> sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 6, captura a *feature* que pertence a um conjunto de *features* alternativas (tipo *ALTERNATIVE*) ou seja obrigatoriamente uma delas deve ser selecionada, que não tem uma delas selecionada.

Quadro 6 – Regra SWRL de validação do produto 3

```
1 SPL:Feature(?f) ^ decisionModel:CurrentState(?f, "↩
     ELIMINATED") ^
2 SPL:hasRelationType(?f, "alternative") -> sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 7, captura a *feature* selecionada pertencente a um conjunto de *features* alternativas, onde alguma outra *feature* também está selecionada.

Quadro 7 – Regra SWRL de validação do produto 4

```
1 SPL:Feature(?f1) ^ SPL:Feature(?f2) ^
2 SPL:alternativeFeatures(?f1, ?f2) ^
3 decisionModel:CurrentState(?f1, "SELECTED") ^
4 decisionModel:CurrentState(?f2, "SELECTED")
5 -> sqwrl:select(?f1)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 8, captura a *feature* que tiver o estado atual igual ao informado no parâmetro "?g".

Quadro 8 – Regra SWRL de validação do produto 5

```
1 SPL:Feature(?f) ^ decisionModel:CurrentState(?f, ?g) ->  ↩
     sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 9, retorna o número de *features* selecionadas .

Quadro 9 – Regra SWRL de validação do produto 6

```
1 SPL:Feature(?f) ^ decisionModel:CurrentState(?f,"SELECTED"↩
     )
2 -> sqwrl:count(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 10, captura a *feature* informada no parâmetro.

Quadro 10 – Regra SWRL de validação do produto 7

```
1 SPL:Feature(?f1) -> sqwrl:select(?f1)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 11, captura o grupo de *features* no qual pelo menos uma *features* deve ser selecionada.

Quadro 11 – Regra SWRL de validação do produto 8

```
1 SPL:FeatureGroup(?f) ^ SPL:hasRelationGroupType(?f, "at-↩
     least-one")
2 -> sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 12, captura o grupo de *features* onde deve ser selecionada apenas uma ou nenhuma *feature* do grupo mas tem pelo menos duas selecionadas.

Quadro 12 – Regra SWRL de validação do produto 9

```
1 SPL:FeatureGroup(?f) ^ SPL:hasRelationGroupType(?f, "zero-↩
     or-one")
2 ^ SPL:hasNodes(?f, ?x1) ^ SPL:hasNodes(?f, ?x2) ^
3 decisionModel:CurrentState(?x1, "SELECTED") ^
4 decisionModel:CurrentState(?x2, "SELECTED") -> ↩
     sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 13, captura o grupo de *features* onde apenas uma delas deve ser selecionada.

Quadro 13 – Regra SWRL de validação do produto 10

```
1 SPL:FeatureGroup(?f) ^ SPL:hasRelationGroupType(?f, "↩
     exactly-one")
2 -> sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

A regra semântica SWRL apresentada no Quadro 14, captura o grupo de *features* onde apenas uma delas deve ser selecionada mas tem pelo menos duas selecionadas.

Quadro 14 – Regra SWRL de validação do produto 11

```
1 SPL:FeatureGroup(?f) ^ SPL:hasRelationGroupType(?f, "↩
     exactly-one")
2 ^ SPL:hasNodes(?f, ?x1) ^ SPL:hasNodes(?f, ?x2) ^
3 decisionModel:CurrentState(?x1, "SELECTED") ^
4 decisionModel:CurrentState(?x2, "SELECTED") -> ↩
     sqwrl:select(?f)
```

Fonte: Autor desta dissertação, 2015.

Vale ressaltar que esse conjunto de regras pode ser alterado de acordo com a necessidade, bastando apenas adicionar, remover ou modificar as regras no diretório de regras (*productRules*) da ferramenta.

4.5.3 Aspectos de Implementação do Componente *Knowledge Base*

Knowledge Base é o componente central da arquitetura. Pois, permite realizar a persistência e manipulação de dados semânticos e relacionais. A Figura 23 apresenta o diagrama de classes da implementação desse componente, onde destaca-se a arquitetura em três camadas, descritas a seguir.

Camada *Controller*

KnowledgeBase é a classe que serve como interface única para manipulação da *Knowledge Base* pelos componentes externos. Agrega as classes *RelationalDatabaseController* e *OntologyController* e provê um acesso simplificado às mesmas.

RelationalDatabaseController classe responsável por gerenciar e repassar as requisições de manipulação dos registros do banco relacional à camada *Model and Persistence*.

OntologyController classe responsável por gerenciar e repassar as requisições de manipulação das ontologias à camada *Model and Persistence*.

Camada *Model and Persistence*

User é uma classe simples que tem apenas atributos referentes aos dados básicos de um usuário da aplicação, métodos *set* para alterar valores dos atributos e métodos *get* para recuperação dos valores dos atributos.

UserDAO é a classe que implementa as operações básicas de manipulação de usuários (*User*) no banco de dados relacional utilizando o *Hibernate*.

OntologyPersistence é a classe que manipula ontologias no repositório de ontologias. Algumas operações realizadas por esta classe são: a adição, remoção, atualização de ontologias no Sesame e a exportação da ontologia do repositório para um arquivo owl.

GenerycKAO classe responsável por manipular as instâncias das ontologias de forma orientada a objetos. Esta classe trabalha de forma semelhante ao padrão de projeto *DAO*, mapeando as chamadas da aplicação com a camada de persistência, realizando operações nas ontologias sem expor seus detalhes e sua complexidade. Um aspecto importante desta classe é que a mesma funciona de forma genérica mudando seu comportamento de acordo com a classe de modelo passada como parâmetro.

SoftwareProductLine é uma classe modelo de uma linha de produtos de software. Esta classe foi implementada utilizando anotações providas pelo GKMT que relacionam seus atributos e métodos a sua respectiva ontologia de forma que quando passada como parâmetro para um objeto do tipo *GenerycKAO* as alterações do estado do objeto podem ser refletidas nas ontologias do repositório.

Node é uma classe modelo de um nó do modelo de *features* da linha de produtos de software. Sua implementação é semelhante à classe *SoftwareProductLine*.

Feature é uma classe modelo de uma *feature* do modelo de *features* da linha de produtos de software. Sua implementação é semelhante à classe *SoftwareProductLine* e é classe filha de *Node*.

FeatureGroup é uma classe modelo de um grupo de *features* do modelo de *features* da linha de produtos de software. Sua implementação é semelhante à classe *SoftwareProductLine* e é classe filha de *Node*.

Learner é uma classe modelo de um aluno do tutor inteligente gerado. Sua implementação é semelhante à classe *SoftwareProductLine*.

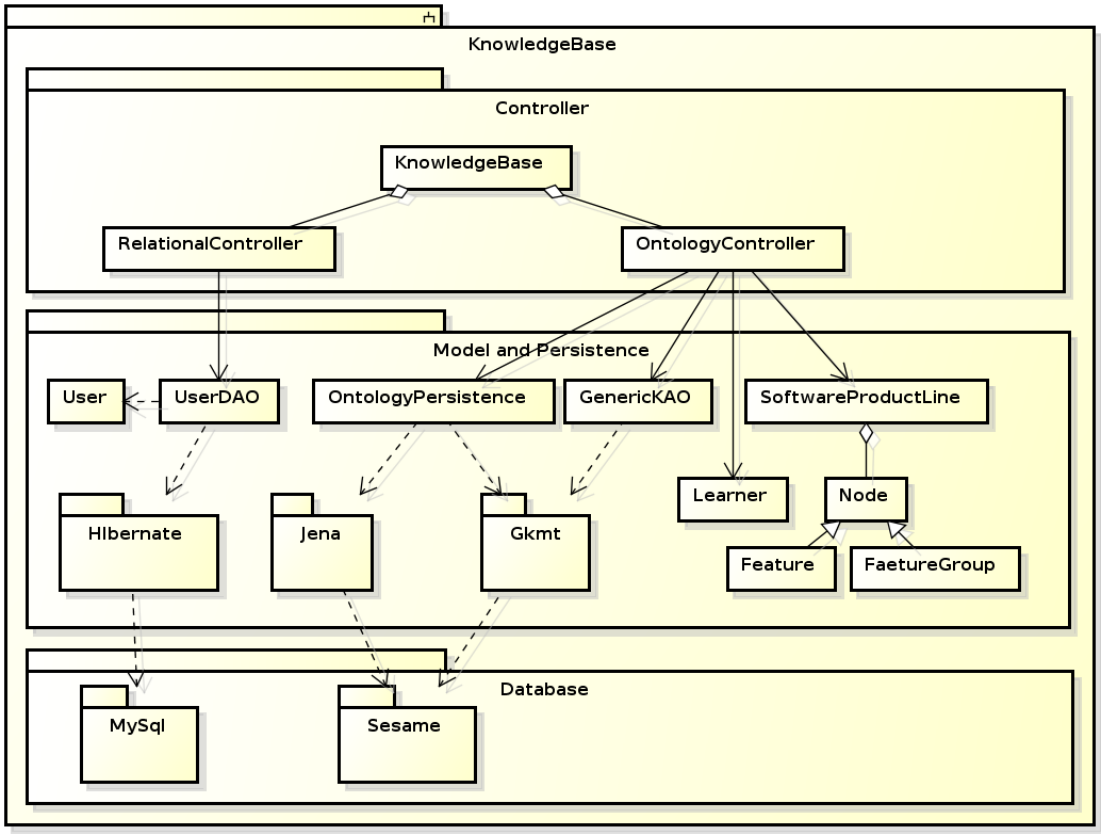
Camada *Database*

MySQL [22] é um sistema gerenciador de banco de dados relacional. Tem a responsabilidade de armazenar os dados relacionais da ferramenta como por exemplo os dados dos usuários.

[22] <https://www.mysql.com>

Sesame [23] é um *framework open source* para inferência, armazenamento e consulta de dados RDF. Tem responsabilidade de armazenar as ontologias da ferramenta.

Figura 23 – Diagrama de classes do componente *KnowlegedBase*



Fonte: Autor desta dissertação, 2015.

4.5.4 Ontologias Utilizadas

As quatro ontologias (i.e., SPL, ITS, Decison Model e Learner) do modelo apresentado na Seção 4.4.1, interagem para que possa ser realizada a descrição semântica da LPS desde a sua definição até a geração do produto no contexto de STIs. Essas ontologias gerais, foram implementadas na linguagem OWL utilizando o software Protégé. Os códigos em OWL destas ontologias são apresentados no Apêndice C e descritos de forma sucinta a seguir.

SPL.owl é a implementação da ontologia que descreve os componentes básicos de uma LPS qualquer (i.e., nó (node), *feature*, grupo de *features* (*FeatureGroup*)). Para que um produto de uma LPS possa ser instanciado deve ser criado uma ontologia derivada desta onde os indivíduos da ontologia descrevam os componentes da LPS específica.

[23] <http://rdf4j.org>

STI.owl é a implementação de uma ontologia derivada de LPS que descreve o modelo de *features* de uma LPS de STIs.

decisonModel.owl é a implementação da ontologia que descreve as decisões tomadas pelo autor/projetista em relação a quais *features* serão incluídas ou removidas do produto a se gerado. Ela faz a ligação entre a ontologia derivada de LPS e outras ontologias referentes ao domínio do produto. No contexto de STIs são as ontologias STI.owl e Learner.owl.

Learner.owl é a implementação da ontologia que descreve o modelo o aluno de um sistema tutor inteligente. Além dessa ontologia, pode-se agregar outras ontologias do domínio do STI.

5 ESTUDO DE CASO E AVALIAÇÃO

Este capítulo apresenta os resultados obtidos com a implementação da solução proposta. A Seção 5.1 apresenta um estudo de caso da utilização de um protótipo de uma ferramenta que implementa o modelo proposto no Capítulo 4 para a instanciação de um STI a partir de uma LPS semântica que utiliza as ontologias apresentadas no Apêndice C. A Seção 5.2 apresenta um estudo a fim de avaliar os benefícios de uma ferramenta que implementa o modelo proposto nesta dissertação. Por fim na Seção 5.3 é apresentada uma discussão sobre o estudo de caso e a avaliação realizada.

5.1 ESTUDO DE CASO

Este estudo de caso apresenta a utilização do modelo proposto para a engenharia de aplicação de uma abordagem de LPS semântica para STIs. Para tanto, utilizou-se um protótipo de uma ferramenta que implementa o modelo proposto no Capítulo 4 em um cenário mínimo de configuração para a instanciação de um STI a partir de uma LPS semântica, que anote semanticamente seu modelo de *features*, o conhecimento do STI e o modelo do estudante de acordo com as ontologias apresentadas no Apêndice C, e dinâmica, ou seja, têm a capacidade de adaptar produtos em tempo de execução de acordo com a ontologia do produto.

Mais especificamente, essa ferramenta foi utilizada na engenharia de aplicação da LPS de STIs proposta por (SILVA, 2011) que baseia-se no *framework* para STIs MASSAYO (PINTO, 2009) desenvolvida pelo antigo grupo de pesquisa GrOW, atual Núcleo de Excelência em Tecnologias Sociais (NEES[1]) da UFAL[2].

Essa ferramenta foi implementada com base em uma LPS dinâmica que permite ser configurada de acordo com um modelo de *features* semântico, anotado de acordo com as ontologias já apresentadas. Depois de anotar semanticamente o modelo de *features* da LPS e o conhecimento do STI utilizando as ontologias, deve-se inseri-las no sistema juntamento com os artefatos de software da LPS (compactados em um arquivo WAR[3]) através do módulo de cadastro de LPS (*SPL Registration*), neste estudo foram cadastradas previamente as ontologias da LPS de STIs proposta por (SILVA, 2011) com o nome de "MASSAYO ITS SPL".

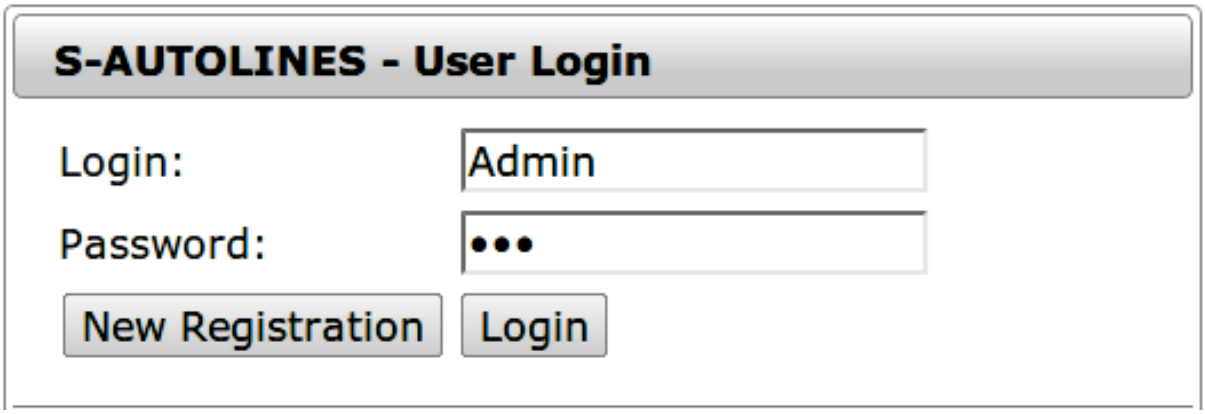
A instanciação de um novo produto de uma LPS inicia-se com a autenticação do autor do STI no sistema através do módulo de autenticação de usuário (*User Login*, ver Figura 24).

[1] <www.nees.com.br>
[2] Universidade Federal de Alagoas - <http://www.ufal.edu.br>
[3] *Web application ARchive*

Neste módulo, o autor informa seu nome de usuário e sua senha, nos campos *Login* e *Password* respectivamente, e clica no botão *Login* para que suas informações sejam autenticadas e seu acesso liberado.

Caso o autor ainda não esteja cadastrado na ferramenta, deve clicar no botão de novo cadastro (*New Registration*) para que seja redirecionado para a tela do módulo de cadastro de usuário (*User Registration*) onde deve preencher os dados requisitados e ao concluir seu cadastro repetir o processo de autenticação descrito anteriormente. Neste estudo de caso, foi utilizado um autor padrão com nome de login ”admin” cadastrado previamente (mostrado na Figura 24).

Figura 24 – Tela do módulo de autenticação do usuário (*User Login*)

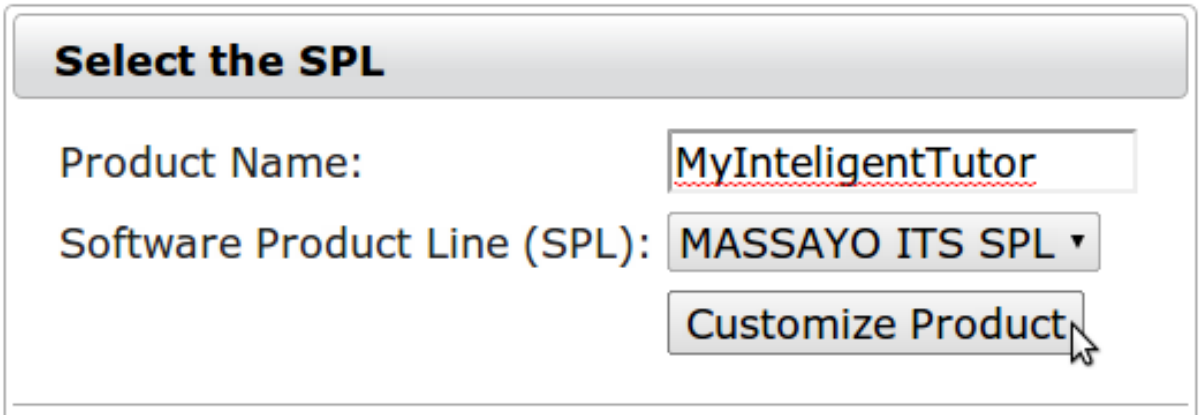


Fonte: Autor desta dissertação, 2015.

O processo de engenharia de aplicação para instanciar um STI de uma LPS é composto, na ferramenta, de cinco passos ou etapas: Seleção da LPS (*SPL Selection*), Customização (ou configuração) do Produto (*Product Customization*), Validação de *Features* (*Feature Validation*), Geração do Produto (*Product Generation*) e Implantação do Produto (*Product Deployment*), descritas a seguir.

Seleção da LPS nessa etapa, o autor deve selecionar a LPS e o nome do produto a ser gerado. Para tanto, o autor deve informar o nome do tutor através do campo Nome do Produto (*Product Name*), escolher a LPS entre as opções existentes da lista Linha de Produtos de Software (*Software Product Line*) e clicar no botão Personalizar Produto (*Customize Product*), como mostrado na Figura 25.

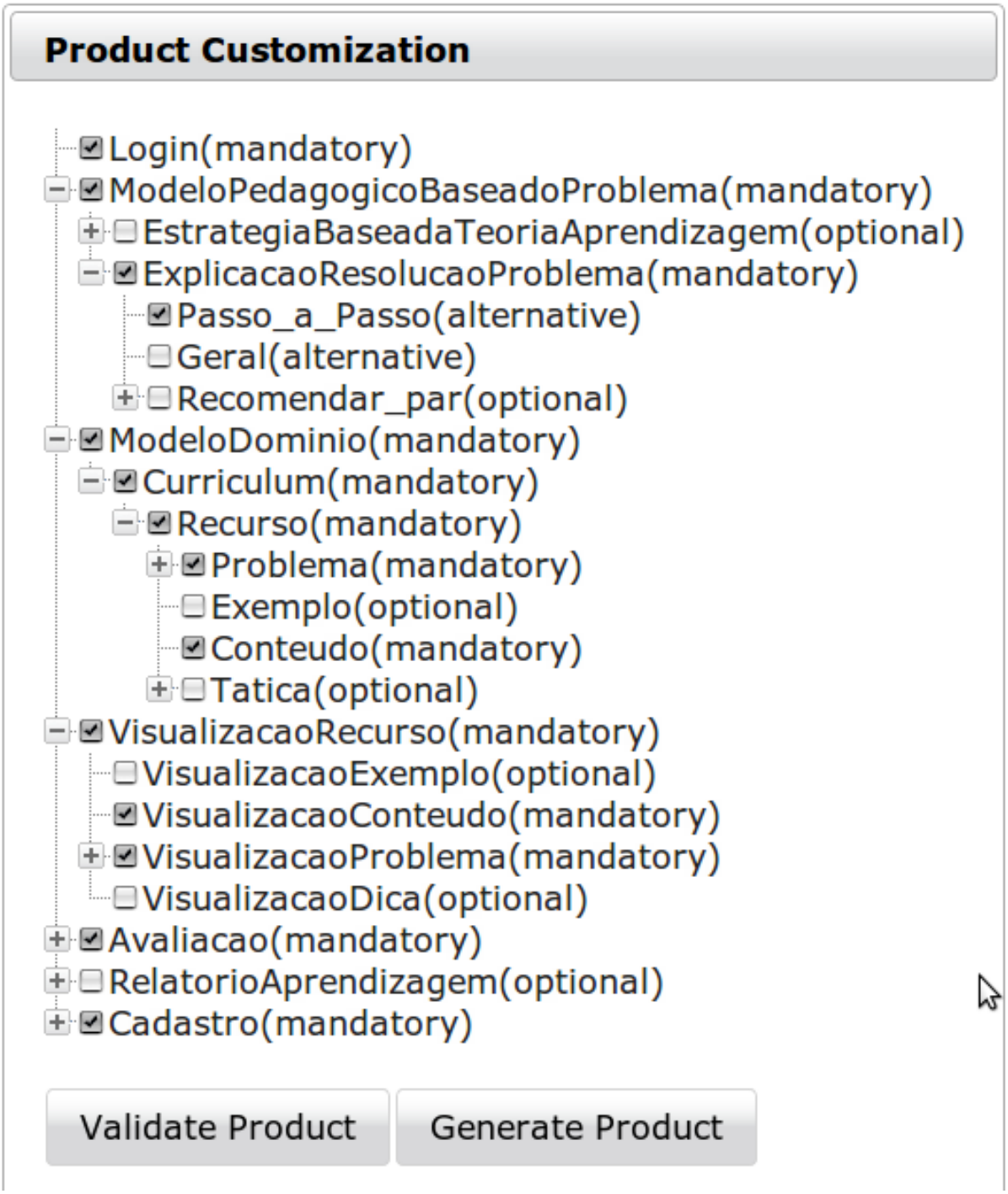
Figura 25 – Etapa de seleção da LPS (*SPL Selection*)



Fonte: Autor desta dissertação, 2015.

Personalização do Produto nessa etapa, o autor deve configurar o seu produto a partir do modelo de *features* da LPS escolhida na etapa de Seleção da LPS. Como ilustrado na Figura 26, deve-se clicar na caixa de seleção no lado esquerdo do nome da *feature* para marca-la caso deseje incluir no seu produto final ou caso a *feature* já esteja selecionada pode-se desmarca-la retirando-a do produto. As *features* podem ser do tipo obrigatória (*mandatory*) que devem ser obrigatoriamente selecionadas, opcional (*optional*) que podem ser ou não selecionadas e alternativa (*alternative*) que caso seja selecionada não deve ser selecionada juntamente com outra *feature* alternativa a ela. O tipo de cada *feature* é apresentado entre parêntese no lado direito de seu nome.

Figura 26 – Etapa de customização do produto (*SPL Customization*)



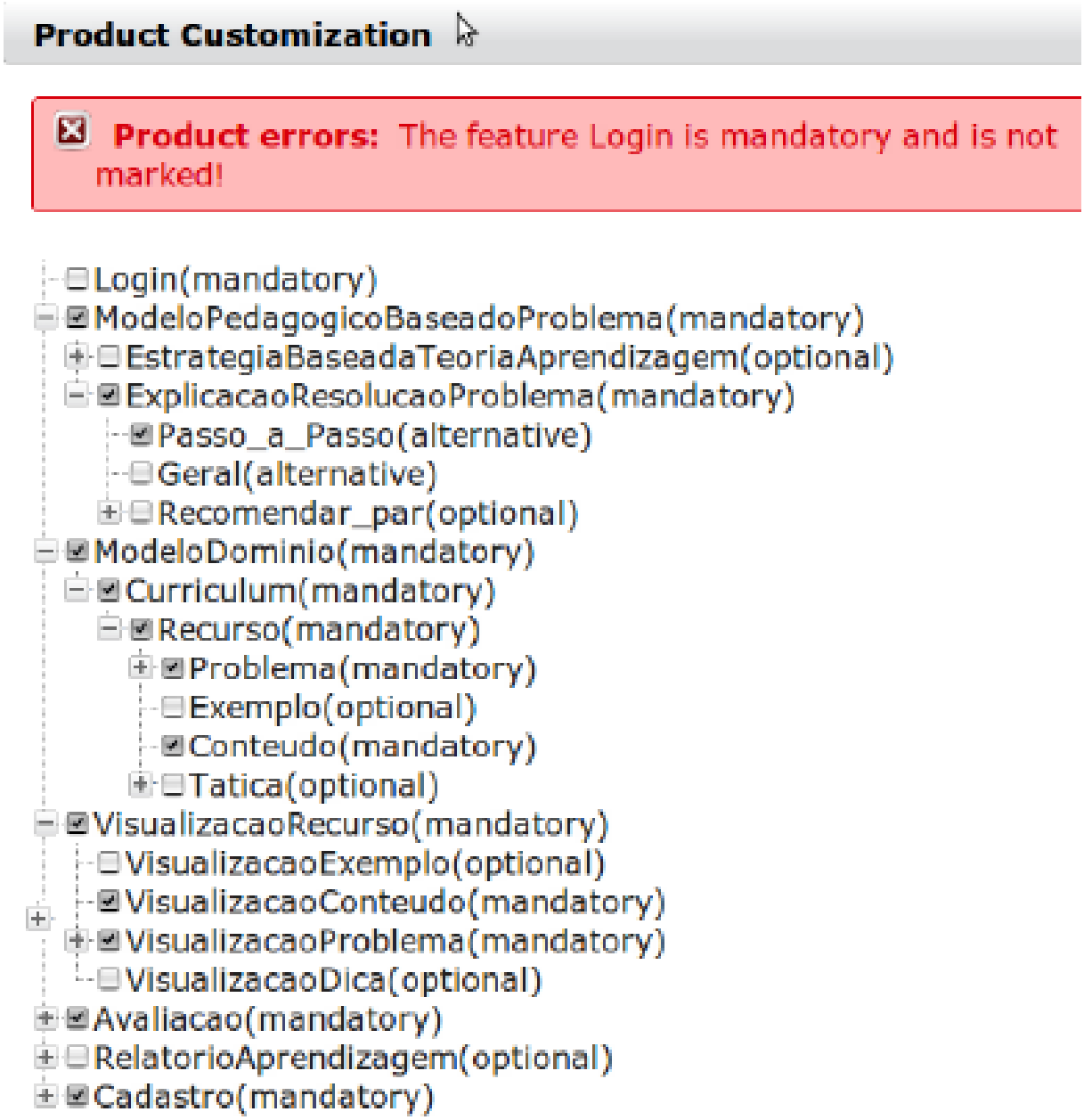
Fonte: Autor desta dissertação, 2015.

Validação de *Features* nessa etapa, o autor deve clicar no botão Validar Produto (*Validate Product*) para que a ferramenta de autoria verifique se há erros na configuração do modelo de *features*.

Caso existam erros, a ferramenta informa ao autor para que sejam corrigidos. Como exemplo, a Figura 27 exibe um erro detectado pela execução da regra 4 que capturou a *feature* obrigatória (*Mandatory*) Login não selecionada (i.e., o valor da propriedade *CurrentState* da ontologia *decisonModel* para Login está "*ELIMINATED*"). Se mais de um erro for detectado, é gerada uma mensagem similar a exibida na Figura 27 contendo a relação de todos os erros detectados.

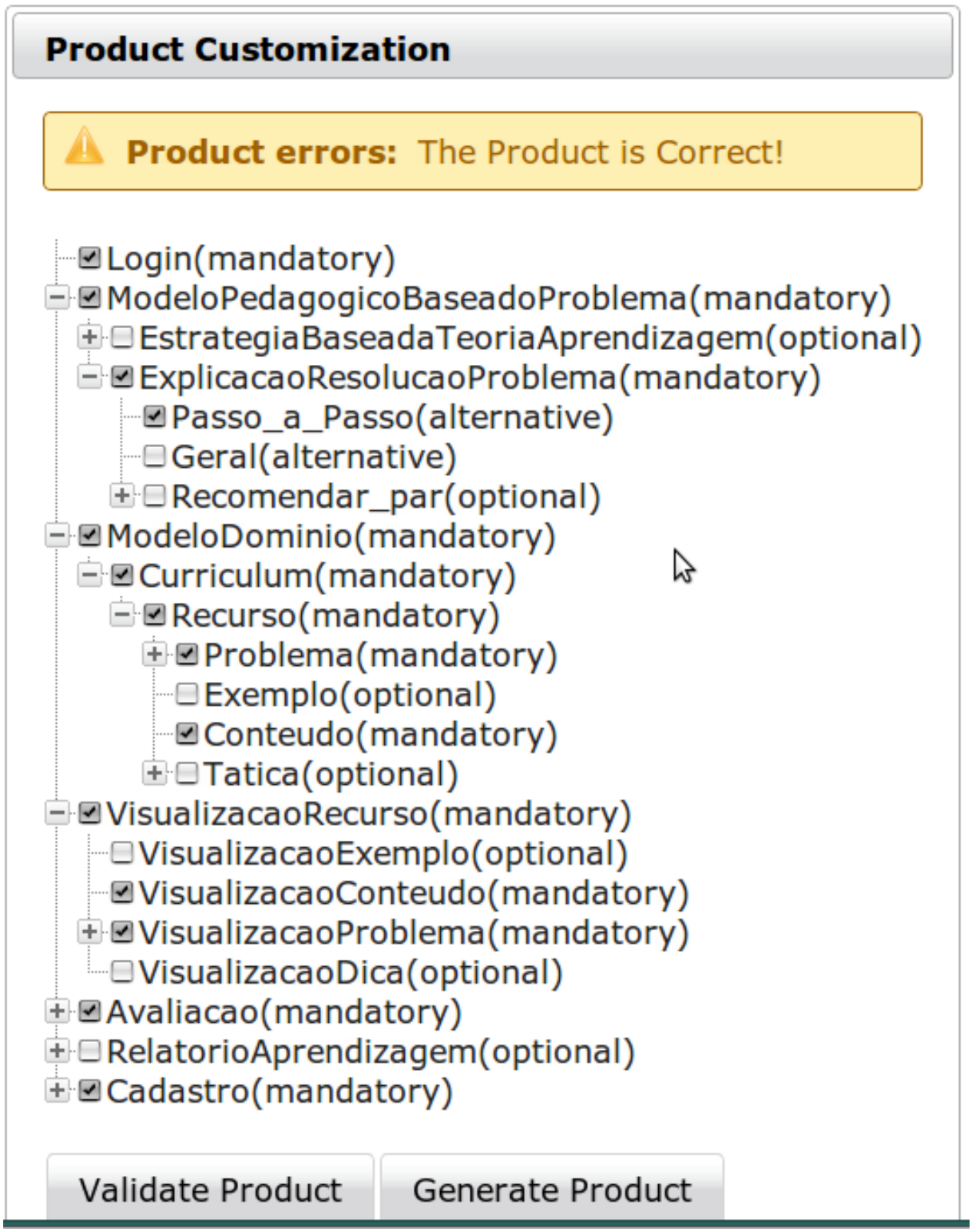
Se nenhuma *feature* foi capturada pelas regras de validação então a ferramenta exibe uma mensagem que o produto está correto (Figura 28) e permite que o autor possa gerar o produto.

Figura 27 – Mensagem de erros encontrados na validação das (*features*)



Fonte: Autor desta dissertação, 2015.

Figura 28 – Mensagem de que o produto está correto (i.e., sem erros)



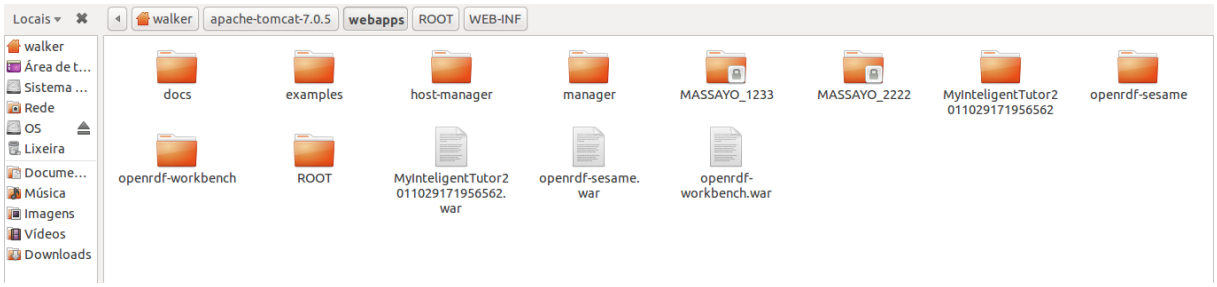
Fonte: Autor desta dissertação, 2015.

Geração do Produto após a validação do modelo de *features* do produto e da exibição da

mensagem que o produto está correto (Figura 28), então, o autor deve clicar no botão Gerar Produto (*Generate Product*) para que a ferramenta de autoria gere o produto customizado. Como resultado a ferramenta irá gerar um arquivo do tipo WAR e um arquivo OWL correspondente ao modelo de decisão produto gerado.

Implantação do Produto essa etapa realizada automaticamente após a geração do produto. Consiste na implantação do produto gerado e suas ontologias em um servidor, definido nas configurações da ferramenta, e a exibição de uma mensagem ao usuário com o endereço de acesso do STI implantado (ver Figura 30) . Nesse estudo de caso, foi utilizado um servidor Apache Tomcat (ver Figura 29).

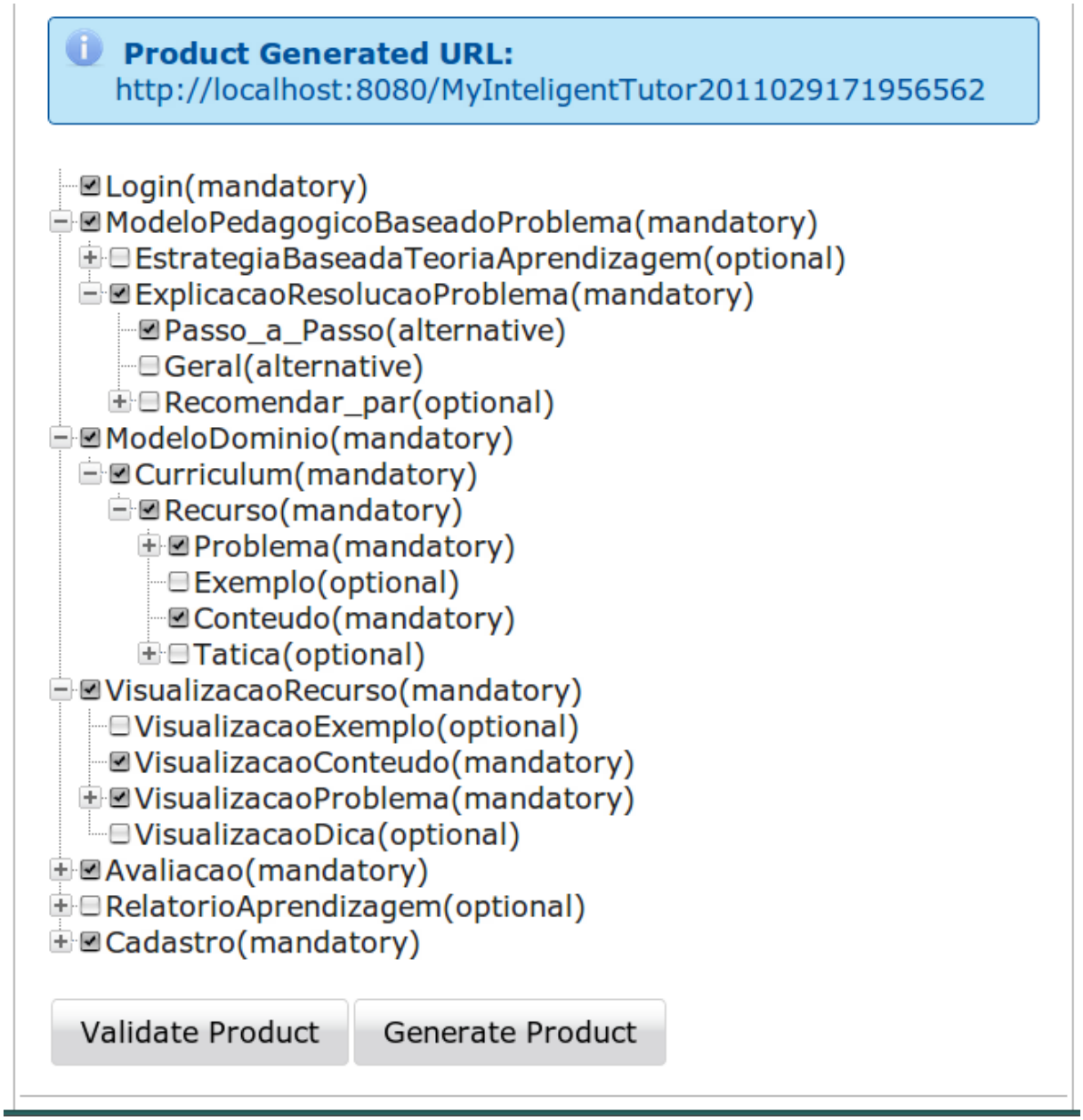
Figura 29 – Pasta *webapps* do servidor Web com o produto instanciado e implantado



Fonte: Autor desta dissertação, 2015.

Após as etapas descritas, é necessário apenas que o autor/projetista acesse a URL informada na mensagem exibida (ver Figura 30) em um navegador de internet para utilizar o STI instanciado (ver Figura 31).

Figura 30 – Mensagem com a URL do produto gerado



Fonte: Autor desta dissertação, 2015.

Figura 31 – STI Web implantado e funcionado

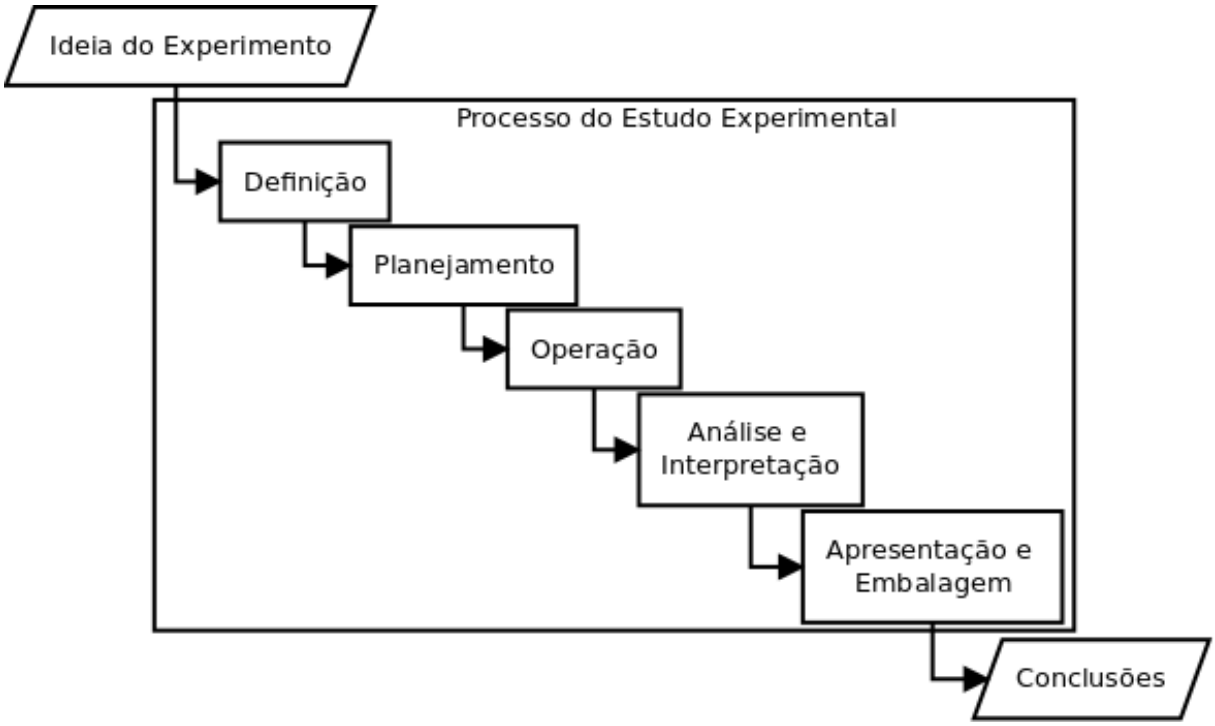
Fonte: Autor desta dissertação, 2015.

5.2 AVALIAÇÃO

Como o objetivo de avaliar o modelo proposto neste trabalho e suas contribuições na construção de STIs em uma abordagem de LPS semântica, foi realizado um estudo da utilização de uma ferramenta que implementa o modelo proposto. Este estudo baseou-se no método de avaliação proposto por (WOHLIN et al., 2000).

De modo geral, o processo de estudo experimental proposto por (WOHLIN et al., 2000) (ver Figura 32) é composto por etapas que buscam levar da ideia do experimento até a obtenção de conclusões. O processo inicia-se com a etapa de definição onde o experimento é explicado em termos do problema abordado, objetivos e métricas utilizadas; na etapa de planejamento é projetado como será realizado o experimento considerando os materiais e métodos a serem utilizados. Também nesta etapa, realiza-se a avaliação das ameaças à experiência. Na etapa de operação, o planejamento do experimento é executado e os dados são coletados para que sejam analisados e interpretados na etapa de análise e interpretação e posteriormente os resultados sejam organizados e apresentados na etapa de apresentação e embalagem.

Figura 32 – Visão geral do processo de estudo experimental



Fonte: Adaptada de (WOHLIN et al., 2000).

Realizadas as considerações iniciais a respeito do processo de estudo experimental utilizado, as próximas linhas desta seção descrevem a aplicação desse processo para a ferramenta, também chamada de SAUTOLines.

A etapa de definição possui três fases: definição do objetivo, definição das questões quantitativas e qualitativas e definição das métricas de avaliação do experimento. Para a ferramenta proposta temos como principal objetivo do estudo:

O1: Analisar a ferramenta que implementa o modelo proposto em relação a sua complexidade

(i.e., conhecimento prévio ou habilidades requeridas do autor), esforço em relação ao tempo gasto para instanciar um STI a partir de uma abordagem de LPS semântica e dinâmica e a precisão da validação das *features* do produto, do ponto de vista dos autores/projetistas.

As questões relevantes levantadas para este estudo de caso são:

Q1: Há redução de esforço em relação ao tempo gasto pelos autores na instanciação de um STI da LPS escolhida? Esta questão quantitativa é observada a fim de analisar se a ferramenta proporciona economia de tempo.

Q2: Há redução de complexidade em relação as habilidades requeridas do autor na instanciação de um STI da LPS escolhida? Para responder esse questionamento os autores foram convidados a descrever as dificuldades encontradas durante a experiência e as competências que foram necessárias.

Q3: Há aumento da eficácia de instanciar um STI com a abordagem de validação realizada na ferramenta? A fim de entender se a validação de *features* fornece recomendações eficazes, o *recall* (necessidade de refazer a validação) e a precisão das recomendações foram analisadas.

Para completar a etapa de definição do estudo foram definidas as métricas, conjuntos de dados associados a cada questão com o intuito de responde-las de forma quantitativa (SANTOS et al., 2012; WOHLIN et al., 2000), a seguir.

M1: Variação de Esforço (VE) na instanciação de STI semântico (BASILI et al., 1996) é calculado utilizando a seguinte equação:

$$VE = \frac{EMFP}{EMSFP} \tag{1}$$

EMFP: Esforço (tempo) médio para autores instanciarem um STI semântico utilizando a ferramenta proposta.

EMSFP: Esforço (tempo) médio para autores instanciarem um STI semântico sem utilizar a ferramenta proposta.

A métrica de variação do esforço compreende o intervalo de [0, N] onde VE 0 indica a redução máxima do esforço em termos de tempo gasto quando se está utilizando a ferramenta proposta e $VE >= 1$ indica que o esforço de instanciar um STI semântico é igual ou superior a não utiliza-la.

M2: Precisão na validação das *features* (P) é calculado utilizando a seguinte equação:

$$P = \frac{VP}{VP + FP} \tag{2}$$

VP: Verdadeiro-Positivo. FP: Falso-Positivo.

A métrica de precisão na validação das *features* compreende o intervalo de [0, 1] onde P 0 indica a falta total de precisão na detecção de erros no modelo de *features* na validação realizada pela ferramenta proposta e $P = 1$ indica a precisão máxima onde todos os erros do modelo de *features* do STI semântico foram corretamente detectados.

M3: *Recall* na validação das *features* é calculado utilizando a seguinte equação:

$$R = \frac{VP}{VP + FN} \tag{3}$$

VP: Verdadeiro-Positivo. FN: Falso-Negativo.

A métrica de *recall* indica a necessidade de realizar novamente a validação do modelo de *features* devido a não detecção de algum erro. Esta métrica é obtida a partir da divisão da quantidade de recomendações corretamente realizadas (VP) pela soma das recomendações corretas e das recomendações que não foram realizadas mas deveriam (FN). O valor de R compreende o intervalo [0, 1] onde 0 indica a necessidade máxima de revalidar o modelo de *features* e $R = 1$ indica que todos os erros do modelo de *features* do STI semântico foram corretamente detectados e não há necessidade de revalida-lo.

Após a definição do experimento, etapa que determina o "porque" do experimento ser realizado, inicia-se o planejamento do experimento que define "como" o experimento será realizado.

O planejamento do experimento inicia-se com o detalhamento do contexto do experimento definido na etapa anterior. Este experimento realizou-se de forma "off-line" em um laboratório de pesquisa. Os sujeitos possuem diferentes níveis de habilidade e foram selecionados pela técnica de amostragem por conveniência[4].

Antes dos sujeitos iniciarem o experimento aplicou-se um questionário de coleta de dados a respeito da escolaridade e conhecimentos anteriores (Apêndice B).

Após o detalhamento do contexto do experimento são definidas as hipóteses do experimento. Para este experimento foram definidas as hipóteses a seguir.

[4] Técnica onde o pesquisador se utiliza dos elementos mais disponíveis da população. Geralmente o autor envia convites aos interessados que se disponibilizam a responder a pesquisa.

Hipótese Nula (H0) A hipótese nula determina que não há benefício na utilização da ferramenta proposta no processo de instanciação de STIs de uma LPS semântica e dinâmica. Mais especificamente não há benefício em utilizar a ferramenta proposta para validar o modelo de *features* nem redução do esforço ou complexidade na tarefa de instanciação.

H0.1 esforço (redução de tempo) com a ferramenta proposta (dado pela métrica M1) > 1

H0.2 precisão da validação do modelo de `features` com a ferramenta proposta (dado pela métrica M2) $< 50\%$

H0.3 *recall* da validação do modelo de `features` com a ferramenta proposta (dado pela métrica M3) $< 50\%$

Hipótese Alternativa (H1) A hipótese alternativa determina que há benefício na utilização da ferramenta proposta no processo de instanciação de STIs de uma LPS semântica e dinâmica. Mais especificamente indica que esta ferramenta tem a capacidade de reduzir o tempo e a complexidade da tarefa de instanciação de STIs e traz benefícios na validação do modelo de *features*.

H1.1 esforço (redução de tempo) com a ferramenta proposta (dado pela métrica M1) $<= 1$

H1.2 precisão da validação do modelo de `features` com a ferramenta proposta (dado pela métrica M2) $>= 50\%$

H1.3 *recall* da validação do modelo de `features` com a ferramenta proposta (dado pela métrica M3) $>= 50\%$

Neste experimento utilizou-se apenas uma variável independente, que pode-se controlar e alterar no experimento, que é a ferramenta proposta utilizada para auxiliar os autores na instanciação dos STIs da LPS semântica e dinâmica. E as variáveis dependentes, que não podem ser controladas e alteradas no experimento, são (a) o esforço (tempo gasto) na instanciação de um STI, (b) a precisão e (c) o *recall* da validação do modelo de *features* da linha.

O planejamento do experimento utiliza um fator (a ferramenta proposta) com dois tratamentos definidos a seguir.

Tratamento 1: no primeiro tratamento, os sujeitos utilizaram a ferramenta proposta para instanciar dois produtos distintos de uma LPS semântica e dinâmica. O tempo gasto e as dificuldades encontradas foram coletados e os resultados apresentados.

Tratamento 2: no segundo tratamento, os sujeitos não utilizaram a ferramenta proposta para instanciar dois produtos distintos de uma LPS semântica e dinâmica. O tempo gasto e as dificuldades encontradas foram coletados e os resultados apresentados.

Para guiar os sujeitos da pesquisa, foi entregue um descrição do experimento contendo informações a seu respeito tais como o objetivo. E para aumentar a precisão dos dados coletados, os participantes do experimento foram orientados a utilizar uma planilha de tempo para anotar o tempo gasto nos tratamentos.

Uma vez que o experimento foi definido e planejado ele deve ser operacionalizado a fim de recolher os dados que devem ser analisados.

Na etapa de operação os tratamentos, definidos na etapa de planejamento do experimento, são aplicados aos sujeitos do experimento. Esta etapa consiste em três tarefas: preparação, onde os sujeitos são escolhidos, informados a respeito do experimento e o material é preparado; execução, onde os sujeitos realizam as tarefas de acordo com os tratamentos planejados e os dados são coletados; validação dos dados, onde os dados coletados são validados (WOHLIN et al., 2000).

Na preparação do experimento foram escolhidos sujeitos através da técnica de amostragem por conveniência. Estes sujeitos são estudantes da Universidade Federal de Alagoas (UFAL) e pesquisadores do NEES. As informações a respeito do experimento foram passadas através de uma apresentação oral e para cada sujeito foi atribuído um computador do laboratório do NEES, entregue os questionários e uma planilha de tempo.

Todos os instrumentos de pesquisa foram preparados e o experimento foi executado em dezembro de 2011 nas dependências do laboratório do NEES na UFAL. Onde inicialmente cada sujeito respondeu ao questionário de avaliação do sujeito (os resultado estão ilustrados na Tabela 1), seguido pela realização da tarefa de instanciação de dois STIs da LPS semântica e dinâmica "MASSAYO ITS SPL"de forma convencional (tratamento 2) e então realizaram a mesma tarefa utilizando a ferramenta proposta (tratamento 1), em ambos os tratamentos os tempos foram registrados pelos sujeitos na planilha de tempo. Por fim, os sujeitos reponderam ao questionário de avaliação da ferramenta proposta (Apêndice A).

Para finalizar a etapa de operação, foram coletados os dados dos cinco sujeitos e como todos participaram do experimento da forma como foram instruídos seus dados são válidos (i.e., podem ser incluídos na pesquisa).

Baseado nos dados coletados na etapa de operação é possível tirar conclusões válidas. Para tanto faz-se necessário interpretar os dados do experimento (WOHLIN et al., 2000).

Tabela 1 – Resultados do questionário dos sujeitos

ID	Semestre na graduação	Complexidade dos projetos	Principal função nos projetos	Experiência com ontologias	Experiência com reuso de software	Experiência com LPS	Já utilizou ferramenta de autoria de STIs?
1	3	Baixa	Desenvolvedor de GUI	Nenhuma	Baixa	Média	Não
2	3	Baixa	Desenvolvedor	Baixa	Baixa	Média	Não
3	3	Baixa	Desenvolvedor	Nenhuma	Baixa	Baixa	Não
4	3	Baixa	Desenvolvedor	Média	Nenhuma	Nenhuma	Não
5	3	Baixa e média	desenvolvedor, acadêmico	Nenhuma	Nenhuma	Baixa	Sim

Fonte: Autor desta dissertação, 2015.

Na etapa análise e interpretação foram analisados os dados quantitativos e qualitativos do experimento e na etapa apresentação e embalagem os resultados do experimento foram apresentados em forma de tabelas. Os resultados quantitativos são apresentados na Tabela 2 onde a coluna ID apresenta o identificador dos sujeitos do experimento, a coluna TT1 representa o tempo gasto no tratamento 1 utilizando a ferramenta proposta, a coluna TT2 representa o tempo gasto no tratamento 2 sem utilizar a ferramenta proposta, a coluna PT1 indica a precisão da validação do modelo de *features* utilizando a ferramenta proposta (i.e., tratamento 1) e PT2 indica a precisão sem utilizar a ferramenta proposta (i.e., tratamento 2).

Tabela 2 – Análise quantitativa do experimento

ID	TT1 (min)	TT2 (min)	PT1 (%)	PT2 (%)
1	5,0	15,0	100,0	79,0
2	4,5	16,3	100,0	85,0
3	3,0	14,0	100,0	72,0
4	5,3	14,7	100,0	63,0
5	3,8	18,0	100,0	69,0

Fonte: Autor desta dissertação, 2015.

Segundo (WOHLIN et al., 2000), a estatística descritiva pode ser utilizada para descrever e apresentar aspectos interessantes de um conjunto de dados. Aplicando estatística descritiva aos dados da Tabela 2 obteve-se a Tabela 3 nela observou-se que utilizando a ferramenta proposta os sujeitos obtiveram um tempo médio de 4,32 minutos, tempo mínimo de 3,0 minutos e tempo máximo de 5,3 minutos na instanciação de um STI. Sem usar a ferramenta, o tempo médio foi 15,4 minutos, com mínimo de 14,0 e máximo de 18,0 minutos. Estes dados indicam que os sujeitos gastam menos tempo na instanciação de STI utilizando a ferramenta proposta do que

sem utiliza-la. Em relação a precisão na configuração válida do modelo de *features* obteve-se 100% de precisão média utilizando a ferramenta (PMT1), enquanto que sem utilizar a ferramenta a precisão (PMT2) na configuração válida do modelo de *features* obteve um valor médio de 73,6% valor mínimo de 63% e máximo de 85%. Isto indica que instanciar produtos semânticos de uma LPS semântica e dinâmica é uma tarefa complexa e que requer conhecimento prévio das tecnologias envolvidas e que o uso da ferramenta proposta facilita a realização tarefa e com o máximo de precisão.

Tabela 3 – Estatística descritiva do experimento

Estatística	EMFP (min)	EMSFP (min)	PMT1 (%)	PMT2 (%)
Média	4,32	15,4	100,0	73,6
Mínimo	3,0	14,0	100,0	63,0
Máximo	5,3	18,0	100,0	85,0

Fonte: Autor desta dissertação, 2015.

É necessário testar as hipóteses com o objetivo de verificar se é possível rejeitar a hipótese nula (H0) e confirmar a hipótese alternativa (H1) (WOHLIN et al., 2000). Após calcular as métricas M1, M2 e M3 definidas na etapa de definição e aplicá-las à hipótese H0, obtemos a Tabela 4, onde verificou-se que todas as três condições foram rejeitadas; enquanto que aplicando as métricas à hipótese H1 obtemos a Tabela 5, onde nenhuma das três condições foi rejeitada e assim pôde-se confirmar a hipótese H1.

Tabela 4 – Teste da Hipótese Nula

	Hipótese Nula	Resultado da Métrica	Rejeitado
H0.1	M1 > 1	0,28	sim
H0.2	M2 $< 50\%$	100%	sim
H0.3	M3 $< 50\%$	80%	sim

Fonte: Autor desta dissertação, 2015.

Tabela 5 – Teste da Hipótese Alternativa

	Hipótese Alternativa	Resultado da Métrica	Rejeitado
H1.1	M1 $<= 1$	0,28	não
H1.2	M2 $>= 50\%$	100%	não
H1.3	M3 $>= 50\%$	80%	não

Fonte: Autor desta dissertação, 2015.

Com o objetivo de avaliar os aspectos qualitativos a repeito da usabilidade, funcionalidade e as impressões dos sujeitos em relação a ferramenta proposta foi aplicado um questionário de avaliação da ferramenta (Apêndice A) composto pelas perguntas a seguir.

P1: Qual a utilidade da ferramenta na atividade de construção de sistemas tutores inteligentes (STIs)?

P2: Qual a utilidade das mensagens apresentadas pela validação das funcionalidades do produto a ser construído.

P3: Você teve algum problema com a validação das funcionalidades do produto?

P4: Se você teve algum problema com a validação das funcionalidades do produto, cite-os.

P5: Na sua opinião, a ferramenta fornece uma interface limpa e útil?

P6: Qual a utilidade do componente de exibição em árvore para verificar as dependências entre as funcionalides do produto?

P7: Você acha que há alguma outra informação importante que deve estar presente na interface da ferramenta? (Se sim citá-los ou então deixe em branco)

P8: Você encontrou algum outro problema que não foi mencionado? Cite-os.

P9: Por favor, escreva qualquer sugestão para melhorar a utilidade da ferramenta.

Nas perguntas P1, P2, P5 e P6, foi atribuído pelo sujeito um valor de 1 a 5, onde 1 representa “Nada útil” e 5 “Muito útil’; nas perguntas P4, P7, P8 e P9, o sujeito pôde responder de forma livre, e na P3 "Sim"ou "Não".

Um dos sujeitos não respondeu às perguntas do questionário de avaliação da ferramenta e assim apenas as respostas dos outros quatro sujeitos foram analisadas.

Os resultados deste questionário são apresentados na Tabela 6 e indicam que a ferramenta proposta possui vários aspectos favoráveis tais como foi útil na atividade de instanciação de STI, exibiu detalhes úteis na validação das *features* do produto e o componente de exibição em árvore da interface gráfica da ferramenta proposta foi útil na verificação das dependências entras as *features* do produto. Também foram dadas contribuições de melhorias, tais como, o visual da

ferramenta pode ser melhorado, pode ser adicionado o suporte ao navegador internet *explorer* e as *features* obrigatórias (i.e., *Mandatory*) já deveriam está marcadas.

Tabela 6 – Resultados do questionário qualitativo a respeito do uso da ferramenta proposta

ID	P1	P2	P3	P4	P5	P6	P7	P8	P9
1	4	4	Não	–	3	3	Não	–	–
2	4	4	Não	–	4	3	Um selecionamento sem ser em cascata. Algo mais "visual"	–	Os "Mandatory" podiam vim marcados
3	5	3	Não	–	5	5	Mais informações sobre cada uma das ferramentas. Talvez dividi-las por seção	–	Campos obrigatórios não deveriam ser citados..
4	4	4	Não	–	3	4	–	–	As opções que são obrigatórias já têm que estar marcadas.
5	–	–	–	–	–	–	–	–	–

Fonte: Autor desta dissertação, 2015.

5.3 DISCUSSÃO

No estudo de caso realizado, foi apresentado o uso da ferramenta que implementa o modelo proposto nessa dissertação em um cenário mínimo de configuração para a instanciação de um STI a partir de uma abordagem de LPS semântica. O que se pôde observar é que a ferramenta que implementa o modelo proposto nessa dissertação tem uma interface gráfica com usuário simples e com poucas telas sequenciais. O que permite guiar o autor/projetista de forma intuitiva nas tarefas da engenharia de aplicação de forma que não seja requerido conhecimentos de engenharia de software e nem de ontologias.

Para avaliar se modelo proposto atende aos objetivos da pesquisa foi realizado um estudo com a ferramenta. Nesse estudo o processo de instanciação de STIs de uma LPS foi realizada por voluntários que não tinham conhecimentos avançados em reuso de software, ontologias, LPS e STIs. De forma preliminar, pôde-se verificar que a ferramenta demonstrou-se de fácil operação reduzindo o esforço e a complexidade empregada na tarefa de instanciação do STI e tornando o processo mais fácil e rápido se comparado a não utiliza-la.

Os resultados na avaliação das métricas de precisão e *recall* da validação de produtos indicaram que a ferramenta provê bons resultados auxiliando de forma significativa os sujeitos

da pesquisa na construção dos produtos.

Nos testes de hipóteses as três condições da hipótese nula foram rejeitadas e as três condições da hipótese alternativa foram aceitas o que confirma que a ferramenta traz benefícios na redução do esforço e da complexidade realizado pelos autores na engenharia de aplicação de um abordagem de LPS semântica.

A avaliação qualitativa indica aspectos favoráveis na utilização da ferramenta que implementa o modelo proposto. Os sujeitos relataram ainda as seguintes contribuições e melhorias: a ferramenta proposta contribui na análise com o componente de exibição em árvore, auxiliando seus usuários no entendimento das dependências entre as *features*; o visual da *ferramenta* pode ser melhorado e pode ser adicionado o suporte ao navegador internet *explorer*.

Como ameaça à validade do experimento, pode-se citar a pequena amostra pequena, os sujeitos eram estudantes de computação, os sujeitos eram do mesmo laboratório do autor desta dissertação e a ordem de execução dos tratamentos.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho está contextualizado na pesquisa de Engenharia de Software na Educação, concentra-se na busca de uma modelo para Engenharia de Aplicação (EA) de uma abordagem de Linha de Produto de Software (LPS) e tecnologias semânticas para construção de Sistemas Tutores Inteligentes (STIs).

Seguindo as orientações dos objetivos, este trabalho propõe um modelo baseado em ontologia para a engenharia de aplicação de LPS para STIs. De maneira mais específica, o modelo proposto torna semiautomático os processos de customização, instanciação e implantação de STIs de uma LPS, tornando transparente ao usuário a manipulação de ontologias e artefatos de software.

O modelo proposto utiliza ontologias para representar as funcionalidades e restrições de uma LPS genérica, as funcionalidades específicas para STIs, as decisões do autor em termos de quais funcionalidades farão parte do STI a ser gerado e as informações do aluno. O processo de engenharia de aplicação da LPS é realizado por componentes que conduzem o autor pelas etapas de autenticação no sistema, seleção da LPS a ser instanciada, customização/configuração das funcionalidades do STI, validação da configuração, geração e implantação de um STI em um servidor Web.

Para validar o modelo proposto foi construída uma ferramenta que implementa o modelo proposto. Foram utilizadas na implementação da ferramenta tecnologias Java Web (e.g., JSF2) e Web semânticas (e.g., OWL e SWRL). Por fim, realizou-se um estudo de caso, com a ferramenta citada, abrangendo a engenharia de aplicação de um STI a partir de uma LPS semântica.

Os resultados obtidos mostram-se adequados apontando como principais contribuições a concepção e desenvolvimento de um modelo semântico para a engenharia de aplicação de LPS para STIs, este modelo guia o autor pelo processo tornando transparente o uso de ontologias e LPS, auxilia na redução da complexidade e do esforço empregado (i.e., carga de trabalho) na construção de STIs a partir de LPS semântica, reduz as qualificações exigidas para instanciar STIs o que pode possibilitar que mais pessoas realizem essa tarefa e permite validar corretamente a configuração das funcionalidades do STI a ser instanciado de forma que apenas produtos sem erros de configuração sejam gerados.

Os resultados deste trabalho foram divulgados nas formas a seguir.

- Apresentação em eventos científicos:
  - Apresentação oral do trabalho "*An ontology-based model for driving the building*

of software product lines in an ITS context" na "Third International Conference on Software, Services and Semantic Technologies S3T 2011";

  - Apresentação oral do trabalho "*A semantic tool to assist authors in the instantiation of software product lines for intelligent tutoring systems context*" na "4th workshop on semantic web and education - WSWEd" em 2011.

- Publicação em veículos científicos:

  - Artigo intitulado: "*An ontology-based model for driving the building of software product lines in an ITS context*" no livro "Third International Conference on Software, Services and Semantic Technologies S3T 2011, Advances in intelligent and soft computing, Vol. 101, Springer Berlin, Heidelberg (2011), pp. 155-159" em 2011. Disponível em <http://dx.doi.org/10.1007/978-3-642-23163-6_22>;
  - Artigo intitulado: "*A semantic tool to assist authors in the instantiation of software product lines for intelligent tutoring systems context*" pelo *journal IEEE Technology and Engineering Education* (ITEE) em 2012. Disponível em <http://itee-edsoccom.com/index.php/itee/article/viewFile/230/241>.

Quanto aos trabalhos futuros da pesquisa apresentada nesta dissertação, destacam-se:

Recomendação de *features* como trabalho futuro, sugere-se estender o modelo proposto de modo que a partir do perfil do autor e do domínio do STI possam ser realizadas recomendações ao autor/projetista de *features* da LPS a serem selecionadas na customização do produto a ser instanciado.

Validação automática a ferramenta implementada utilizando o modelo proposto, não provê na interface com o usuário correções do modelo de *features* ao mesmo tempo em que o autor está realizando a seleção das *features* que farão parte do STI a ser instanciado. Isto decorre do tempo gasto no processo de validação através de regras semânticas, o que prejudicaria a usabilidade do sistema caso fosse constantemente realizada. Como trabalho futuro, sugere-se melhorar o desempenho do mecanismo de validação do modelo de *features* de forma que possa ser realizada a validação da configuração do STI ao mesmo tempo que o usuário está realizando a customização do mesmo e sem que para isto necessite de um *hardware* muito robusto.

Evolução de STIs existentes como trabalho futuro, sugere-se adicionar à interface com o usuário da ferramenta implementada um mecanismo voltado a reconfiguração de STIs existentes com base em novas funcionalidades adicionadas à LPS.

REFERÊNCIAS

ALEVEN, V.; MCLAREN, B.; SEWALL, J.; KOEDINGER, K. The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In: IKEDA, M.; ASHLEY, K.; CHAN, T.-W. (Ed.). **Intelligent Tutoring Systems**. Springer Berlin Heidelberg, 2006, (Lecture Notes in Computer Science, v. 4053). p. 61–70. ISBN 978-3-540-35159-7. Disponível em: <http://dx.doi.org/10.1007/11774303_7>.

ALEVEN, V.; SEWALL, J.; POPESCU, O.; VELSEN, M. van; DEMI, S.; LEBER, B. Reflecting on twelve years of its authoring tools research with ctat. **Design Recommendations for Intelligent Tutoring Systems: Authoring Tools and Expert Modeling Techniques**, Robert Sottilare, p. 263, 2015.

ALMEIDA M; BAX, M. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. **Revista Ciência da Informação**, 2003.

ANDERSON, J. R. **Rules of the mind**. [S.l.]: Psychology Press, 2014.

ARPÍREZ, J. C.; CORCHO, O.; FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A. Webode: a scalable workbench for ontological engineering. In: ACM. **Proceedings of the 1st international conference on Knowledge capture**. [S.l.], 2001. p. 6–13.

BASILI, V. R.; BRIAND, L. C.; MELO, W. L. How reuse influences productivity in object-oriented systems. **Communications of the ACM**, v. 39, n. 10, p. 104–116, 1996.

BENJAMIN, N.; XIANGEN, H.; GRAESSER, A.; ZHIQIANG, C. Autotutor in the cloud: A service-oriented paradigm for an interoperable natural-language its. **Journal of Advanced Distributed Learning Technology**, v. 2, n. 6, p. pp49–63, 2014.

BITTENCOURT, I. I.; BRITO, P.; PEDRO, A.; ISOTANI, S.; JAQUES, P. A.; RUBIRA, C. Desafios da engenharia de software na educação: Variabilidade de sistemas educacionais inteligentes e instanciação em larga escala. In: **Anais do Workshop de Desafios da Computação Aplicada à Educação**. [S.l.: s.n.], 2012. p. 70–79.

BITTENCOURT, I. I.; COSTA, E.; SILVA, M.; SOARES, E. A computational model for developing semantic web-based educational systems. **Knowledge-Based Systems**, Elsevier, Amsterdam, The Netherlands, v. 22, n. 4, p. 302–315, 2009.

BITTENCOURT, I. I.; ISOTANI, S.; COSTA, E.; MIZOGUCHI, R. Research directions on semantic web and education. **Scientia - Interdisciplinary Studies in Computer Science**, Citeseer, v. 19, n. 1, p. 60–67, 2008.

BLOOM, B. S. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. **Educational Researcher**, American Educational Research Association, v. 13, n. 6, p. pp. 4–16, 1984. ISSN 0013189X. Disponível em: <http://www.jstor.org/stable/1175554>.

BORST, W. N. **Construction of Engineering Ontologies for Knowledge Sharing and Reuse**. Tese (PhD thesis) — University of Twente, P.O. Box 217 - 7500 AE Enschede - The Nether-lands, 1997.

BULL, S.; KAY, J. Student models that invite the learner in: The smili open learner modelling framework. **International Journal of Artificial Intelligence in Education**, IOS Press,

v. 17, n. 2, p. 89–120, 2007. Disponível em: <http://www.eee.bham.ac.uk/bull/papers-pdf/ijaied07-smili.pdf>.

BULL, S.; KAY, J. Open learner models. In: NKAMBOU, R.; BOURDEAU, J.; MIZOGUCHI, R. (Ed.). **Advances in Intelligent Tutoring Systems**. Springer Berlin Heidelberg, 2010, (Studies in Computational Intelligence, v. 308). p. 301–322. ISBN 978-3-642-14362-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-14363-2_15>.

BULL, S. Q. S.; MABBOTT, A. Computer-based formative assessment to promote reflection and learner autonomy. **Electronic, Electrical and Computer Engineering, University of Birmingham, UK**, v. 1, 2006. Disponível em: <http://www.eee.bham.ac.uk/bull/papers-pdf/EngEd-06.pdf>.

BURNS, H.; LUCKHARDT, C. A.; PARLETT, J. W.; REDFIELD, C. L. **Intelligent tutoring systems: Evolutions in design**. [S.l.]: Psychology Press, 2014.

CALERO, C.; RUIZ, F.; PIATTINI, M. **Ontologies for Software Engineering and Software Technology**. 1st. ed. Berlin, Heidelberg: Springer Publishing Company, Incorporated, 2010. ISBN 3642070876, 9783642070877.

CAPILLA, R.; BOSCH, J.; KANG, K.-C. **Systems and Software Variability Management**. Heidelberg, NY, USA: Springer, 2013.

CARDOSO, J.; PINTO, A. M. The web ontology language (owl) and its applications. **Encyclopedia of Social Network Analysis and Mining**, 2015. Disponível em: <http://jorge-cardoso.github.io/publications/Papers/BC-2015-031-ISR-OWL-and-Its-Applications.pdf>.

CLEMENTS, P. C.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. (SEI Series in Software Engineering). ISBN 0-201-70332-7.

COHEN, M. B.; DWYER, M. B.; SHI, J. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. **Software Engineering, IEEE Transactions on**, IEEE, v. 34, n. 5, p. 633–650, 2008.

CORCHO, O.; FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A. Methodologies, tools and languages for building ontologies. where is their meeting point? **Data & knowledge engineering**, Elsevier, v. 46, n. 1, p. 41–64, 2003.

COSTA, F. A. Para uma definição de metas de aprendizagem na área das tic em portugal. **Revista e-Curriculum**, Pontifícia Universidade Católica de São Paulo, v. 7, n. 1, p. p. 2–12, April 2011.

COUNCIL, T. **How People Learn: Brain, Mind, Experience, and School, expanded ed**. Washington, DC, USA: National Academy Press, Washington, DC, 2000. ISBN 0-309-07036-8. Disponível em: <http://www.nap.edu/read/9853/chapter/1>.

CZARNECKI, K.; HELSEN, S.; EISENECKER, U. W. Staged configuration through specialization and multilevel configuration of feature models. **Software Process: Improvement and Practice**, v. 10, n. 2, p. 143–169, 2005.

DECKER, S.; ERDMANN, M.; FENSEL, D.; STUDER, R. **Ontobroker: Ontology based access to distributed and semi-structured information**. [S.l.]: Springer, 1999.

DHUNGANA, D.; RABISER, R.; GRüNBACHER, P.; NEUMAYER, T. Integrated tool support for software product line engineering. In: **Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering**. New York, NY, USA: ACM, 2007. (ASE '07), p. 533–534. ISBN 978-1-59593-882-4. Disponível em: <http://doi.acm.org/10.1145/1321631.1321730>.

FARQUHAR, A.; FIKES, R.; RICE, J. The ontolingua server: A tool for collaborative ontology construction. **International journal of human-computer studies**, Elsevier, v. 46, n. 6, p. 707–727, 1997.

FENSEL, D.; HARMELEN, F. V.; HORROCKS, I.; MCGUINNESS, D. L.; PATEL-SCHNEIDER, P. F. Oil: An ontology infrastructure for the semantic web. **Intelligent Systems, IEEE**, IEEE, v. 16, n. 2, p. 38–45, 2001.

FERNANDES, O. D. d. C. R. **Construção e exploração de ontologias para a negociação automática em empresas virtuais**. Tese (Doutorado) — Universidade do Porto, 2007.

FIGUEIREDO, E.; CACHO, N.; SANT'ANNA, C.; MONTEIRO, M.; KULESZA, U.; GARCIA, A.; SOARES, S.; FERRARI, F.; KHAN, S.; DANTAS, F. et al. Evolving software product lines with aspects. In: IEEE. **Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on**. Leipzig, Germany, 2008. p. 261–270.

FOWLER, D. G. A model for designing intelligent tutoring systems. **Journal of medical systems**, Springer, v. 15, n. 1, p. 47–63, 1991.

FU, J. S. Ict in education: A critical literature review and its implications. **International Journal of Education and Development using Information and Communication Technology**, University of the West Indies, v. 9, n. 1, p. 112, 2013.

GAMBOA, H.; FRED, A. Designing intelligent tutoring systems: a bayesian approach. **Enterprise Information Systems III. Edited by J. Filipe, B. Sharp, and P. Miranda.**, Springer-Verlag New York, Inc., p. 146–152, 2002.

GAŠEVIĆ, D.; DJURIĆ, D.; DEVEDŽIĆ, V. **Model driven architecture and ontology development**. [S.l.]: Springer Science & Business Media, 2006.

GENESERETH, M. R.; FIKES, R. E. et al. Knowledge interchange format-version 3.0: reference manual. **Stanford University**, Computer Science Department, Stanford University, Stanford, California, USA, 1992.

GILLAIN, J.; FAULKNER, S.; HEYMANS, P.; JURETA, I.; SNOECK, M. Product portfolio scope optimization based on features and goals. In: ACM. **Proceedings of the 16th International Software Product Line Conference**. Salvador, Brazil, 2012. v. 1, p. 161–170.

GIRARDI, R.; LEITE, A. A knowledge-based tool for multi-agent domain engineering. **Know.-Based Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 21, n. 7, p. 604–611, out. 2008. ISSN 0950-7051. Disponível em: <http://dx.doi.org/10.1016/j.knosys.2008.03.036>.

GOMEZ-PEREZ, A.; CORCHO-GARCIA, O.; FERNANDEZ-LOPEZ, M. **Ontological Engineering**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN 1852335513.

GóMEZ-PéREZ MARIANO FERNáNDEZ-LóPEZ, O. C. A. Ontological engineering with examples from the areas of knowledge management, e-commerce and the semantic web. **Springer-Verlag**, New York, USA, 2004.

GRUBER, T. R. **Ontolingua: A mechanism to support portable ontologies**. [S.l.]: Stanford University, Knowledge Systems Laboratory, 1992.

GRUBER, T. R. A translation approach to portable ontology specifications. **Knowledge acquisition**, Elsevier, v. 5, n. 2, p. 199–220, 1993.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. **International journal of human-computer studies**, Academic Press, Inc., Duluth, MN, USA, v. 43, n. 5-6, p. 907–928, dez. 1995. ISSN 1071-5819. Disponível em: <http://dx.doi.org/10.1006/ijhc.1995.1081>.

GRUBER, T. R. **What is an ontology**. 2005. Disponível em: <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>.

GUARINO, N. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In: **International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology**. London, UK, UK: Springer-Verlag, 1997. (SCIE '97), p. 139–170. ISBN 3-540-63438-X. Disponível em: <http://dl.acm.org/citation.cfm?id=645856.669803>.

GUARINO, N. Understanding, building and using ontologies. **Int. J. Hum.-Comput. Stud.**, Academic Press, Inc., Duluth, MN, USA, v. 46, n. 2-3, p. 293–310, mar. 1997. ISSN 1071-5819. Disponível em: <http://dx.doi.org/10.1006/ijhc.1996.0091>.

GUIZZARDI, G. **Desenvolvimento para e com reuso: Um estudo de caso no domínio de Vídeo sob Demanda**. Tese (Doutorado) — Universidade Federal do Espírito Santo, 2000.

HEFLIN, J.; HENDLER, J. **Searching the Web with SHOE**. [S.l.]: Defense Technical Information Center, 2000.

HELFERICH, A.; SCHMID, K.; HERZWURM, G. Product management for software product lines: an unsolved problem? **Communications of the ACM**, ACM, v. 49, n. 12, p. 66–67, 2006.

HENDLER, J.; MCGUINNESS, D. L. The darpa agent markup language. **IEEE Intelligent systems**, v. 15, n. 6, p. 67–73, 2000.

HOLANDA, A. Buarque de. Dicionário aurélio básico da língua portuguesa. **Goiânia: Nova Fronteira/O Popular**, 1997.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; DEAN, M. et al. Swrl: A semantic web rule language combining owl and ruleml. **W3C Member submission**, v. 21, p. 79, 2004.

INEP. Censo da educação superior no brasil. **Brasília: Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira**, 2013. Disponível em: <http://download.inep.gov.br/educacao_superior/censo_superior/apresentacao/2014/coletiva_censo_superior_2013.pdf>.

JANOTA, M.; BOTTERWECK, G. Formal approach to integrating feature and architecture models. In: **Fundamental Approaches to Software Engineering**. Budapest, Hungary: Springer, 2008. p. 31–45.

JASPER, R.; USCHOLD, M. et al. A framework for understanding and classifying ontology applications. In: **Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW**. Seattle, USA: [s.n.], 1999. v. 99, p. 16–21.

JOHANSEN, M. F.; HAUGEN, O.; FLEUREY, F.; ELDEGARD, A. G.; SYVERSEN, T. Generating better partial covering arrays by modeling weights on sub-product lines. In: **Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems**. Berlin, Heidelberg: Springer-Verlag, 2012. (MODELS'12), p. 269–284. ISBN 978-3-642-33665-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-33666-9_18>.

JOHN, I.; EISENBARTH, M. A decade of scoping: a survey. In: CARNEGIE MELLON UNIVERSITY. **Proceedings of the 13th International Software Product Line Conference**. San Francisco, USA, 2009. p. 31–40.

JONES, D.; BENCH-CAPON, T.; VISSER, P. Methodologies for ontology development. In: CITESEER. **Proc. IT&KNOWS Conference of the 15th IFIP World Computer Congress**. [S.l.], 1998. p. 20–35.

JOVANOVIC, J.; GAŠEVIĆ, D.; DEVEDŽIĆ, V. Tangram for personalized learning using the semantic web technologies. **Journal of emerging technologies in web intelligence**, v. 1, n. 1, p. 6–21, 2009.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. **Feature-Oriented Domain Analysis (FODA) Feasibility Study**. [S.l.], 1990.

KANG VIJAYAN SUGUMARAN, S. P. K. C. **Applied Software Product Line Engineering**. Boston, MA, USA: CRC Press, 2010.

KARP, P. D.; CHAUDHRI, V. K.; THOMERE, J. **XOL: An XML-based ontology exchange language**. [S.l.]: July, 1999.

KIFER, M.; LAUSEN, G.; WU, J. Logical foundations of object-oriented and frame-based languages. **Journal of the ACM (JACM)**, ACM, v. 42, n. 4, p. 741–843, 1995.

KOEDINGER, K. R.; ALEVEN, V.; HEFFERNAN, N.; MCLAREN, B.; HOCKENBERRY, M. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In: SPRINGER. **Intelligent Tutoring Systems**. [S.l.], 2004. p. 162–174.

KOEDINGER, K. R.; ANDERSON, J. R.; HADLEY, W. H.; MARK, M. A. et al. Intelligent tutoring goes to school in the big city. **International Journal of Artificial Intelligence in Education (IJAIED)**, v. 8, p. 30–43, 1997.

KULIK, J. A.; FLETCHER, J. Effectiveness of intelligent tutoring systems a meta-analytic review. **Review of Educational Research**, SAGE Publications, 2015.

LANE, H. C.; CORE, M. G.; HAYS, M. J.; AUERBACH, D.; ROSENBERG, M. Situated pedagogical authoring: Authoring intelligent tutors from a studentís perspective. In: SPRINGER. **Artificial Intelligence in Education**. [S.l.], 2015. p. 195–204.

LEUNG, N. K. **Towards an ontology-based knowledge management: An ontology mediation framework to reconcile inter-organizational knowledge**. Tese (Doutorado) — School of Information Systems and Technology, University of Wollongong, 2012. Disponível em: <http://ro.uow.edu.au/theses/3498>.

LINDEN, F. J. v. d.; SCHMID, K.; ROMMES, E. **Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN 3540714367.

LUZ, E. L.; DUTRA, A.; QUISPE-CONDORI, S.; SOUZA, F. N.; FREITAS, F. Avaliação de um programa de formação para integração das tecnologias na educação. **CIAIQ2015**, v. 2, 2015.

MACGREGOR, R. M. Inside the loom description classifier. **ACM Sigart Bulletin**, ACM, v. 2, n. 3, p. 88–92, 1991.

MARCZAL, D. **FARMA: Uma ferramenta de autoria para objetos de aprendizagem de conceitos matemáticos**. Tese (Doutorado) — Universidade Federal do Paraná, 2014.

MASSEN, T. Von der; LICHTER, H. Requiline: A requirements engineering tool for software product lines. **Lecture notes in computer science**, Springer, p. 168–180, 2004.

MCBRIDE, B. The resource description framework (rdf) and its vocabulary description language rdfs. In: **Handbook on ontologies**. [S.l.]: Springer, 2004. p. 51–65.

MCGUINNESS, D. L.; FIKES, R.; HENDLER, J.; STEIN, L. A. Daml+ oil: an ontology language for the semantic web. **Intelligent Systems, IEEE**, IEEE, v. 17, n. 5, p. 72–80, 2002.

MCGUINNESS, D. L.; HARMELEN, F. V. et al. Owl web ontology language overview. **W3C recommendation**, v. 10, n. 10, p. 2004, 2004.

MERINO, P. J. M.; KLOOS, C. D. An architecture for combining semantic web techniques with intelligent tutoring systems. In: SPRINGER. **ITS '08: Proceedings of the 9th international conference on Intelligent Tutoring Systems**. [S.l.], 2008. p. 540–550.

METZGER, A.; POHL, K. Software product line engineering and variability management: achievements and challenges. In: ACM. **Proceedings of the on Future of Software Engineering**. Hyderabad, India, 2014. p. 70–84.

MITROVIC, A.; MARTIN, B.; SURAWEERA, P.; ZAKHAROV, K.; MILIK, N.; HOLLAND, J.; MCGUIGAN, N. Aspire: an authoring system and deployment environment for constraint-based tutors. University of Canterbury. Computer Science and Software Engineering, 2009.

MORAIS, N. S.; POMBO, L.; BATISTA, J.; MOREIRA, A.; RAMOS, F. Uma revisão de literatura sobre o uso das tecnologias da comunicação no ensino superior. **Revista PRISMA. COM**, n. 24, 2014.

MOTTA, E. **Reusable components for knowledge modelling: Case studies in parametric design problem solving**. [S.l.]: IOS press, 1999. v. 53.

MURRAY, T. Authoring intelligent tutoring systems: An analysis of the state of the art. **International Journal of Artificial Intelligence in Education (IJAIED)**, v. 10, p. 98–129, 1999.

MURRAY, T. An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In: **Authoring tools for advanced technology learning environments**. [S.l.]: Springer, 2003. p. 491–544.

MUSEN, M. A. The protÉgÉ project: A look back and a look forward. **AI Matters**, ACM, New York, NY, USA, v. 1, n. 4, p. 4–12, jun. 2015. ISSN 2372-3483. Disponível em: <http://doi.acm.org/10.1145/2757001.2757003>.

NITTO, E. D.; GHEZZI, C.; METZGER, A.; PAPAZOGLOU, M.; POHL, K. A journey to highly dynamic, self-adaptive service-based applications. **Automated Software Engineering**, Springer, v. 15, n. 3-4, p. 313–341, 2008.

PERROUIN, G.; OSTER, S.; SEN, S.; KLEIN, J.; BAUDRY, B.; TRAON, Y. L. Pairwise testing for software product lines: comparison of two approaches. **Software Quality**, Springer, v. 20, n. 3-4, p. 605–643, 2012.

PINTO, I. I. B. S. **Modelos e Ferramentas para a Construção de Sistemas Educacionais Adaptativos e Semânticos**. Tese (Tese de Doutorado) — Universidade Federal de Campina Grande, Campina Grande, 2009.

POHL, K.; BöCKLE, G.; LINDEN, F. J. v. d. **Software Product Line Engineering: Foundations, Principles and Techniques**. 1. ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. Hardcover. ISBN 3540243720.

POHL, K.; METZGER, A. Software product line testing. **Communications of the ACM**, ACM, v. 49, n. 12, p. 78–81, 2006.

POLSON, M. C.; RICHARDSON, J. J. **Foundations of intelligent tutoring systems**. [S.l.]: Psychology Press, 2013.

POULIN, J. S. **Measuring Software Reuse: Principles, Practices, and Economic Models**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996. ISBN 0-201-63413-9.

RODRIGUES, M.; NOVAIS, P.; SANTOS, M. F. Future challenges in intelligent tutoring systems : a framework. In: **International Conference on Multimedia and Information and Communication Technologies in Education**. [S.l.]: Formatex, 2005.

RUSK, J. J.; GASEVIC, D. Semantic web services-based reasoning in the design of software product lines. In: **SPLC (2)**. [S.l.: s.n.], 2008. p. 123–130.

SALDÍAS, G. M. J. C. et al. **Metodologia para a Construção de Interfaces Adaptáveis em Sistemas Tutores Inteligentes.** Tese (Tese de Doutorado) — Instituto de Engenharia Biomédica, UFSC., 2002.

SANTOS, W. B.; ALMEIDA, E. S. D.; MEIRA, S. R. de L. Tirt: A traceability information retrieval tool for software product lines projects. In: IEEE. **Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on**. [S.l.], 2012. p. 93–100.

SEI. **Software Engineering Institute**. 2015. Disponível em: <http://www.sei.cmu.edu/productlines/.>

SELF, J. The defining characteristics of intelligent tutoring systems research: Itss care , precisely. **International Journal of Artificial Intelligence in Education (IJAIED)**, v. 10, p. 350–364, 1999. Disponível em: <https://telearn.archives-ouvertes.fr/hal-00197346/file/self99.pdf>.

SILVA, A. P. da. **Uma Linha de Produto de Software baseada na Web Semântica para Sistemas Tutores Inteligentes**. Tese (Doutorado) — Universidade Federal de Campina Grande, UFCG, Brasil, 2011. Disponível em: <http://docs.computacao.ufcg.edu.br/posgraduacao/teses/2011/Tese_AlanPedrodaSilva.pdf>.

SOTTILARE, R. Special report: Adaptive intelligent tutoring system (its) research in support of the army learning model-research outline. **Army Research Laboratory (ARL-SR-0284)**, 2013.

SOTTILARE, R.; GILBERT, S. Considerations for tutoring, cognitive modeling, authoring and interaction design in serious games. In: **Authoring Simulation and Game-based Intelligent Tutoring workshop at the Artificial Intelligence in Education Conference (AIED) 2011**. [S.l.: s.n.], 2011.

SOTTILARE, R.; GRAESSER, A.; HU, X.; BRAWNER, K. **Design Recommendations for Intelligent Tutoring Systems: Authoring Tools and Expert Modeling Techniques**. [S.l.]: Robert Sottilare, 2015.

SOTTILARE, R. A.; BRAWNER, K. W.; GOLDBERG, B. S.; HOLDEN, H. K. **The generalized intelligent framework for tutoring (GIFT)**. [S.l.]: Orlando, FL: US Army Research Laboratory–Human Research & Engineering Directorate (ARL-HRED), 2012.

STAAB, S.; STUDER, R.; SCHNURR, H.-P.; SURE, Y. Knowledge processes and ontologies. **IEEE Intelligent systems**, IEEE, n. 1, p. 26–34, 2001.

SURE, Y.; ERDMANN, M.; ANGELE, J.; STAAB, S.; STUDER, R.; WENKE, D. **OntoEdit: Collaborative ontology development for the semantic web**. [S.l.]: Springer, 2002.

VANLEHN, K. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. **Educational Psychologist**, v. 46, n. 4, p. 197–221, 2011. Disponível em: <http://dx.doi.org/10.1080/00461520.2011.611369>.

WEISS, D. M.; LAI, C. T. R. **Software Product-line Engineering: A Family-based Software Development Process**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0-201-69438-7.

WENGER, E. Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communications of knowledge. **Los Altos, CA: Morgam Kaufmann Publishers**, 1987.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering: an introduction.** [S.l.]: Kluwer Academic Publishers, 2000.

WOOLF, B. P. **Building Intelligent Interactive Tutors Student-centered strategies for revolutionizing e-learning**. [S.l.]: Morgan Kaufmann Publishers, 2009.

APÊNDICE A – QUESTIONÁRIO PARA A ANÁLISE QUALITATIVA

Questionário para a análise qualitativa da ferramenta

Este questionário registra as impressões dos sujeitos da pesquisa após o uso da ferramenta.

*Obrigatório

1. **Qual a utilidade da ferramenta na atividade de construção de sistemas tutores inteligentes (STIs)?** *
   Marcar apenas uma oval.

	1	2	3	4	5	
Nada útil	◯	◯	◯	◯	◯	Muito útil

2. **Qual a utilidade das mensagens apresentadas pela validação das funcionalidades do produto a ser construído.** *
   Marcar apenas uma oval.

	1	2	3	4	5	
Nada útil	◯	◯	◯	◯	◯	Muito útil

3. **Você teve algum problema com a validação das funcionalidades do produto?** *
   Marcar apenas uma oval.

   ◯ Sim
   ◯ Não

4. **Se você teve algum problema com a validação das funcionalidades do produto, cite-os.**

   .....

5. **Na sua opinião, a ferramenta fornece uma interface limpa e útil?** *
   Marcar apenas uma oval.

	1	2	3	4	5	
Não provê	◯	◯	◯	◯	◯	Provê perfeitamente

6. **Qual a utilidade do componente de exibição em árvore para verificar as dependências entre as funcionalides do produto?** *

Marcar apenas uma oval.

	1	2	3	4	5	
Nada útil	◯	◯	◯	◯	◯	Muito útil

7. **Você acha que há alguma outra informação importante que deve estar presente na interface da ferramenta? (Se sim citá-los ou então deixe em branco)**

.....

8. **Você encontrou algum outro problema que não foi mencionado? Cite-os.**

.....

9. **Por favor, escreva qualquer sugestão para melhorar a utilidade da ferramenta.**

.....

APÊNDICE B – QUESTIONÁRIO SOBRE O PERFIL DOS SUJEITOS

Questionário sobre o perfil dos sujeitos

*Obrigatório

1. **Qual é o seu semestre de graduação (apenas número)?** *

   .....

2. **Qual a complexidade dos projetos de software que você já participou de acordo com as seguintes categorias?**
   Marque todas que se aplicam.

   - [ ] Baixa complexidade
   - [ ] Média complexidade
   - [ ] Alta complexidade

3. **No caso de você já ter participado de algum projeto de software, qual foi o seu papel principal?**

   .....

   .....

   .....

   .....

   .....

4. **Como você define a sua experiência com ontologias?** *
   Marcar apenas uma oval.

   - ◯ Nenhuma
   - ◯ Baixa
   - ◯ Média
   - ◯ Alta

5. **Como você define a sua experiência com reuso de software?** *
   Marcar apenas uma oval.

   - ◯ Nenhuma
   - ◯ Baixa
   - ◯ Média
   - ◯ Alta

6. **Como você define a sua experiência com linha de produtos de software?** *
   Marcar apenas uma oval.

   - ◯ Nenhuma
   - ◯ Baixa
   - ◯ Média
   - ◯ Alta

7. **Você já utilizou alguma ferramenta de autoria para construir sistemas tutores inteligentes (STIs)?** *
   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

APÊNDICE C – ONTOLOGIAS UTILIZADAS

Este apêndice apresenta as ontologias utilizadas neste trabalho. Cada seção apresenta o código em OWL de um ontologia utilizada para implementar a anotação semântica do modelo proposta.

C.1 ONTOLOGIA SPL

```
1  <?xml version="1.0"?>
2  <rdf:RDF
3      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4      xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#↩
         "
5      xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
6      xmlns:owl="http://www.w3.org/2002/07/owl#"
7      xmlns="http://www.nees.com.br/ontologies/sautolines/LPS.owl#"
8      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
9      xmlns:swrl="http://www.w3.org/2003/11/swrl#"
10     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
11     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
12   xml:base="http://www.nees.com.br/ontologies/sautolines/SPL.owl">
13   <owl:Ontology rdf:about=""/>
14   <owl:Class rdf:ID="Feature">
15     <rdfs:subClassOf>
16       <owl:Class rdf:ID="Node"/>
17     </rdfs:subClassOf>
18   </owl:Class>
19   <owl:Class rdf:ID="SoftwareProductLine"/>
20   <owl:Class rdf:ID="FeatureGroup">
21     <rdfs:subClassOf rdf:resource="#Node"/>
22   </owl:Class>
23   <owl:ObjectProperty rdf:ID="hasNodes">
24     <rdfs:range rdf:resource="#Node"/>
25     <rdfs:domain>
26       <owl:Class>
27         <owl:unionOf rdf:parseType="Collection">
28           <owl:Class rdf:about="#FeatureGroup"/>
29           <owl:Class rdf:about="#SoftwareProductLine"/>
30         </owl:unionOf>
31       </owl:Class>
32     </rdfs:domain>
```

```
33    </owl:ObjectProperty>
34    <owl:DatatypeProperty rdf:ID="hasRelationType">
35      <rdfs:range>
36        <owl:DataRange>
37          <owl:oneOf rdf:parseType="Resource">
38            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema↩
                  #string"
39            >mandatory</rdf:first>
40            <rdf:rest rdf:parseType="Resource">
41              <rdf:rest rdf:parseType="Resource">
42                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-↩
                     rdf-syntax-ns#nil"/>
43                <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                     XMLSchema#string"
44                >alternative</rdf:first>
45              </rdf:rest>
46              <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                    XMLSchema#string"
47              >optional</rdf:first>
48            </rdf:rest>
49          </owl:oneOf>
50        </owl:DataRange>
51      </rdfs:range>
52      <rdfs:domain rdf:resource="#Feature"/>
53      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
54    </owl:DatatypeProperty>
55    <owl:DatatypeProperty rdf:ID="hasRelationGroupType">
56      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
57      <rdfs:domain rdf:resource="#FeatureGroup"/>
58      <rdfs:range>
59        <owl:DataRange>
60          <owl:oneOf rdf:parseType="Resource">
61            <rdf:rest rdf:parseType="Resource">
62              <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                    XMLSchema#string"
63              >zero-or-one</rdf:first>
64              <rdf:rest rdf:parseType="Resource">
65                <rdf:rest rdf:parseType="Resource">
66                  <rdf:rest rdf:resource="http://www.w3.org↩
                       /1999/02/22-rdf-syntax-ns#nil"/>
```

```
67                  <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
68                  >zero-or-more</rdf:first>
69                </rdf:rest>
70                <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                     XMLSchema#string"
71                >at-least-one</rdf:first>
72              </rdf:rest>
73            </rdf:rest>
74            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema↩
                 #string"
75            >exactly-one</rdf:first>
76          </owl:oneOf>
77        </owl:DataRange>
78      </rdfs:range>
79    </owl:DatatypeProperty>
80    <owl:SymmetricProperty rdf:ID="alternativeFeatures">
81      <rdfs:range rdf:resource="#Feature"/>
82      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          ObjectProperty"/>
83      <rdfs:domain rdf:resource="#Feature"/>
84      <owl:inverseOf rdf:resource="#alternativeFeatures"/>
85    </owl:SymmetricProperty>
86    <owl:FunctionalProperty rdf:ID="name">
87      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
88      <rdfs:domain>
89        <owl:Class>
90          <owl:unionOf rdf:parseType="Collection">
91            <owl:Class rdf:about="#Feature"/>
92            <owl:Class rdf:about="#SoftwareProductLine"/>
93            <owl:Class rdf:about="#FeatureGroup"/>
94          </owl:unionOf>
95        </owl:Class>
96      </rdfs:domain>
97      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          DatatypeProperty"/>
98    </owl:FunctionalProperty>
99    <owl:FunctionalProperty rdf:ID="description">
100     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          DatatypeProperty"/>
```

```
101     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
102     <rdfs:domain rdf:resource="#SoftwareProductLine"/>
103   </owl:FunctionalProperty>
104   <owl:FunctionalProperty rdf:ID="isChildOf">
105     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           ObjectProperty"/>
106     <rdfs:range rdf:resource="#Node"/>
107     <owl:inverseOf>
108       <owl:InverseFunctionalProperty rdf:ID="isParentOf"/>
109     </owl:inverseOf>
110     <rdfs:domain rdf:resource="#Node"/>
111   </owl:FunctionalProperty>
112   <owl:InverseFunctionalProperty rdf:about="#isParentOf">
113     <rdfs:range rdf:resource="#Node"/>
114     <owl:inverseOf rdf:resource="#isChildOf"/>
115     <rdfs:domain rdf:resource="#Node"/>
116     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           ObjectProperty"/>
117   </owl:InverseFunctionalProperty>
118   <owl:DataRange>
119     <owl:oneOf rdf:parseType="Resource">
120       <rdf:rest rdf:parseType="Resource">
121         <rdf:rest rdf:parseType="Resource">
122           <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema↩
                #string"
123           >UNDECIDED</rdf:first>
124           <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-↩
                syntax-ns#nil"/>
125         </rdf:rest>
126         <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
              string"
127         >ELIMINATED</rdf:first>
128       </rdf:rest>
129       <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
             string"
130       >SELECTED</rdf:first>
131     </owl:oneOf>
132   </owl:DataRange>
133 </rdf:RDF>
```

C.2 ONTOLOGIA DECISION MODEL

```
1  <?xml version="1.0"?>
2  <rdf:RDF
3      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4      xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#↩
           "
5      xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
6      xmlns:Learner="http://www.nees.com.br/ontologies/sautolines/↩
           Learner.owl#"
7      xmlns:owl="http://www.w3.org/2002/07/owl#"
8      xmlns:SPL="http://www.nees.com.br/ontologies/sautolines/SPL.owl↩
           #"
9      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
10     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
11     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
12     xmlns="http://www.nees.com.br/ontologies/sautolines/↩
           decisionModel.owl#"
13     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
14   xml:base="http://www.nees.com.br/ontologies/sautolines/↩
        decisionModel.owl">
15   <owl:Ontology rdf:about="">
16     <owl:imports rdf:resource="http://www.nees.com.br/ontologies/↩
           sautolines/Learner.owl"/>
17     <owl:imports rdf:resource="http://www.nees.com.br/ontologies/↩
           sautolines/SPL.owl"/>
18   </owl:Ontology>
19   <owl:ObjectProperty rdf:ID="hasUsers">
20     <rdfs:range rdf:resource="http://www.nees.com.br/ontologies/↩
           sautolines/Learner.owl#User"/>
21     <rdfs:domain rdf:resource="http://www.nees.com.br/ontologies/↩
           sautolines/SPL.owl#Node"/>
22     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           InverseFunctionalProperty"/>
23   </owl:ObjectProperty>
24   <owl:DatatypeProperty rdf:ID="ownerName">
25     <rdfs:domain rdf:resource="http://www.nees.com.br/ontologies/↩
           sautolines/SPL.owl#SoftwareProductLine"/>
26     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
27     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
```

```
28    </owl:DatatypeProperty>
29    <owl:FunctionalProperty rdf:ID="CurrentState">
30      <rdfs:range>
31        <owl:DataRange>
32          <owl:oneOf rdf:parseType="Resource">
33            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema↩
                  #string"
34            >SELECTED</rdf:first>
35            <rdf:rest rdf:parseType="Resource">
36              <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                    XMLSchema#string"
37              >ELIMINATED</rdf:first>
38              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-↩
                    rdf-syntax-ns#nil"/>
39            </rdf:rest>
40          </owl:oneOf>
41        </owl:DataRange>
42      </rdfs:range>
43      <rdfs:domain rdf:resource="http://www.nees.com.br/ontologies/↩
          sautolines/SPL.owl#Node"/>
44      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          DatatypeProperty"/>
45    </owl:FunctionalProperty>
46  </rdf:RDF>
47
48  <!-- Created with Protege (with OWL Plugin 3.4.4, Build 579)  http:↩
       //protege.stanford.edu -->
```

C.3 ONTOLOGIA ITS

```
1   <?xml version="1.0"?>
2   <rdf:RDF
3       xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#↩
            "
4       xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
5       xmlns:swrlx="http://swrl.stanford.edu/ontologies/built-ins/3.3/↩
            swrlx.owl#"
6       xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
7       xmlns:temporal="http://swrl.stanford.edu/ontologies/built-ins↩
            /3.3/temporal.owl#"
```

```
8       xmlns:tbox="http://swrl.stanford.edu/ontologies/built-ins/3.3/↩
           tbox.owl#"
9       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
10      xmlns:owl="http://www.w3.org/2002/07/owl#"
11      xmlns:SPL="http://www.nees.com.br/ontologies/sautolines/SPL.owl↩
           #"
12      xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins↩
           /3.4/sqwrl.owl#"
13      xmlns:abox="http://swrl.stanford.edu/ontologies/built-ins/3.3/↩
           abox.owl#"
14      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
15      xmlns:swrl="http://www.w3.org/2003/11/swrl#"
16      xmlns="http://www.nees.com.br/ontologies/sautolines/ITS.owl#"
17      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
18      xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#↩
           "
19    xml:base="http://www.nees.com.br/ontologies/sautolines/ITS.owl">
20    <owl:Ontology rdf:about="">
21      <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies↩
           /3.3/swrla.owl"/>
22      <owl:imports rdf:resource="http://www.w3.org/2003/11/swrlb"/>
23      <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/↩
           built-ins/3.3/swrlx.owl"/>
24      <owl:imports rdf:resource="http://www.w3.org/2003/11/swrl"/>
25      <owl:imports rdf:resource="http://www.nees.com.br/ontologies/↩
           sautolines/SPL.owl"/>
26      <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/↩
           built-ins/3.3/abox.owl"/>
27      <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/↩
           built-ins/3.3/temporal.owl"/>
28      <owl:imports rdf:resource="http://sqwrl.stanford.edu/ontologies↩
           /built-ins/3.4/sqwrl.owl"/>
29      <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/↩
           built-ins/3.3/tbox.owl"/>
30    </owl:Ontology>
31    <owl:ObjectProperty rdf:about="http://www.w3.org/2003/11/swrl#↩
         argument2"/>
32    <SPL:SoftwareProductLine rdf:ID="SoftwareProductLine_ITS">
33      <SPL:hasNodes>
34        <SPL:Feature rdf:ID="Login">
35          <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
               string"
```

```
36          >Login</SPL:name>
37          <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
               XMLSchema#string"
38          >mandatory</SPL:hasRelationType>
39        </SPL:Feature>
40      </SPL:hasNodes>
41      <SPL:hasNodes>
42        <SPL:Feature rdf:ID="Professor">
43          <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
               string"
44          >Professor</SPL:name>
45          <SPL:isChildOf>
46            <SPL:Feature rdf:ID="Recomendar_par">
47              <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                   /2001/XMLSchema#string"
48              >optional</SPL:hasRelationType>
49              <SPL:isParentOf>
50                <SPL:Feature rdf:ID="Tutor">
51                  <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
52                  >Tutor</SPL:name>
53                  <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
54                  >optional</SPL:hasRelationType>
55                  <SPL:isChildOf rdf:resource="#Recomendar_par"/>
56                </SPL:Feature>
57              </SPL:isParentOf>
58              <SPL:isParentOf>
59                <SPL:Feature rdf:ID="Estudante">
60                  <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
61                  >optional</SPL:hasRelationType>
62                  <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
63                  >Estudante</SPL:name>
64                  <SPL:isChildOf rdf:resource="#Recomendar_par"/>
65                </SPL:Feature>
66              </SPL:isParentOf>
67              <SPL:isParentOf rdf:resource="#Professor"/>
68              <SPL:isChildOf>
69                <SPL:Feature rdf:ID="ExplicacaoResolucaoProblema">
```

```
70                  <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
71                  >Explicacao da Resolucao do Problema</SPL:name>
72                  <SPL:isChildOf>
73                    <SPL:Feature rdf:ID="↩
                         ModeloPedagogicoBaseadoProblema">
74                      <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                           XMLSchema#string"
75                      >Modelo Pedagogico Baseado em Problema</↩
                           SPL:name>
76                      <SPL:hasRelationType rdf:datatype="http://www.↩
                           w3.org/2001/XMLSchema#string"
77                      >mandatory</SPL:hasRelationType>
78                      <SPL:isParentOf>
79                        <SPL:Feature rdf:ID="↩
                             EstrategiaBaseadaTeoriaAprendizagem">
80                          <SPL:hasRelationType rdf:datatype="http://↩
                               www.w3.org/2001/XMLSchema#string"
81                          >optional</SPL:hasRelationType>
82                          <SPL:isParentOf>
83                            <SPL:Feature rdf:ID="Socratica">
84                              <SPL:name rdf:datatype="http://www.w3.↩
                                   org/2001/XMLSchema#string"
85                              >Socratica</SPL:name>
86                              <SPL:hasRelationType rdf:datatype=
87                              "http://www.w3.org/2001/XMLSchema#↩
                                   string"
88                              >optional</SPL:hasRelationType>
89                              <SPL:isChildOf rdf:resource="#↩
                                   EstrategiaBaseadaTeoriaAprendizagem"↩
                                   />
90                            </SPL:Feature>
91                          </SPL:isParentOf>
92                          <SPL:isParentOf>
93                            <SPL:Feature rdf:ID="Situada">
94                              <SPL:isChildOf rdf:resource="#↩
                                   EstrategiaBaseadaTeoriaAprendizagem"↩
                                   />
95                              <SPL:name rdf:datatype="http://www.w3.↩
                                   org/2001/XMLSchema#string"
96                              >Situada</SPL:name>
97                              <SPL:hasRelationType rdf:datatype=
```

```
98                    "http://www.w3.org/2001/XMLSchema#↩
                         string"
99                    >optional</SPL:hasRelationType>
100                 </SPL:Feature>
101               </SPL:isParentOf>
102               <SPL:isParentOf>
103                 <SPL:Feature rdf:ID="Comportamentalista">
104                   <SPL:isChildOf rdf:resource="#↩
                         EstrategiaBaseadaTeoriaAprendizagem"↩
                         />
105                   <SPL:hasRelationType rdf:datatype=
106                   "http://www.w3.org/2001/XMLSchema#↩
                         string"
107                   >optional</SPL:hasRelationType>
108                   <SPL:name rdf:datatype="http://www.w3.↩
                         org/2001/XMLSchema#string"
109                   >Comportamentalista</SPL:name>
110                 </SPL:Feature>
111               </SPL:isParentOf>
112               <SPL:isParentOf>
113                 <SPL:Feature rdf:ID="Cognitiva">
114                   <SPL:hasRelationType rdf:datatype=
115                   "http://www.w3.org/2001/XMLSchema#↩
                         string"
116                   >optional</SPL:hasRelationType>
117                   <SPL:name rdf:datatype="http://www.w3.↩
                         org/2001/XMLSchema#string"
118                   >Cognitiva</SPL:name>
119                   <SPL:isChildOf rdf:resource="#↩
                         EstrategiaBaseadaTeoriaAprendizagem"↩
                         />
120                 </SPL:Feature>
121               </SPL:isParentOf>
122               <SPL:isParentOf>
123                 <SPL:Feature rdf:ID="Construtivista">
124                   <SPL:hasRelationType rdf:datatype=
125                   "http://www.w3.org/2001/XMLSchema#↩
                         string"
126                   >optional</SPL:hasRelationType>
127                   <SPL:isChildOf rdf:resource="#↩
                         EstrategiaBaseadaTeoriaAprendizagem"↩
                         />
```

```
128                         <SPL:name rdf:datatype="http://www.w3.↩
                               org/2001/XMLSchema#string"
129                         >Construtivista</SPL:name>
130                       </SPL:Feature>
131                     </SPL:isParentOf>
132                     <SPL:isChildOf rdf:resource="#↩
                          ModeloPedagogicoBaseadoProblema"/>
133                     <SPL:name rdf:datatype="http://www.w3.org↩
                          /2001/XMLSchema#string"
134                     >Estrategia baseada em Teoria de ↩
                          Aprendizagem</SPL:name>
135                   </SPL:Feature>
136                 </SPL:isParentOf>
137                 <SPL:isParentOf rdf:resource="#↩
                      ExplicacaoResolucaoProblema"/>
138               </SPL:Feature>
139             </SPL:isChildOf>
140             <SPL:isParentOf>
141               <SPL:Feature rdf:ID="Passo_a_Passo">
142                 <SPL:isChildOf rdf:resource="#↩
                      ExplicacaoResolucaoProblema"/>
143                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                      XMLSchema#string"
144                 >Passo a Passo</SPL:name>
145                 <SPL:alternativeFeatures>
146                   <SPL:Feature rdf:ID="Geral">
147                     <SPL:alternativeFeatures rdf:resource="#↩
                          Passo_a_Passo"/>
148                     <SPL:hasRelationType rdf:datatype="http://↩
                          www.w3.org/2001/XMLSchema#string"
149                     >alternative</SPL:hasRelationType>
150                     <SPL:name rdf:datatype="http://www.w3.org↩
                          /2001/XMLSchema#string"
151                     >Geral</SPL:name>
152                     <SPL:isChildOf rdf:resource="#↩
                          ExplicacaoResolucaoProblema"/>
153                   </SPL:Feature>
154                 </SPL:alternativeFeatures>
155                 <SPL:hasRelationType rdf:datatype="http://www.↩
                      w3.org/2001/XMLSchema#string"
156                 >alternative</SPL:hasRelationType>
157               </SPL:Feature>
```

```
158                 </SPL:isParentOf>
159                 <SPL:isParentOf rdf:resource="#Geral"/>
160                 <SPL:isParentOf rdf:resource="#Recomendar_par"/>
161                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
162                 >mandatory</SPL:hasRelationType>
163               </SPL:Feature>
164             </SPL:isChildOf>
165             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                    XMLSchema#string"
166             >Recomendar_par</SPL:name>
167           </SPL:Feature>
168         </SPL:isChildOf>
169         <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
               XMLSchema#string"
170         >optional</SPL:hasRelationType>
171       </SPL:Feature>
172     </SPL:hasNodes>
173     <SPL:hasNodes>
174       <SPL:Feature rdf:ID="Wiki">
175         <SPL:isChildOf>
176           <SPL:Feature rdf:ID="Tatica">
177             <SPL:isParentOf>
178               <SPL:Feature rdf:ID="Dica">
179                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
180                 >optional</SPL:hasRelationType>
181                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
182                 >Dica</SPL:name>
183                 <SPL:isChildOf rdf:resource="#Tatica"/>
184               </SPL:Feature>
185             </SPL:isParentOf>
186             <SPL:isParentOf rdf:resource="#Wiki"/>
187             <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                    /2001/XMLSchema#string"
188             >optional</SPL:hasRelationType>
189             <SPL:isChildOf>
190               <SPL:Feature rdf:ID="Recurso">
191                 <SPL:isParentOf>
192                   <SPL:Feature rdf:ID="Problema">
```

```
193                 <SPL:hasRelationType rdf:datatype="http://www.↩
                       w3.org/2001/XMLSchema#string"
194                 >mandatory</SPL:hasRelationType>
195                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
196                 >Problema</SPL:name>
197                 <SPL:isChildOf rdf:resource="#Recurso"/>
198                 <SPL:isParentOf>
199                   <SPL:Feature rdf:ID="ProblemaPreencherLacunas↩
                         ">
200                     <SPL:hasRelationType rdf:datatype="http://↩
                           www.w3.org/2001/XMLSchema#string"
201                     >optional</SPL:hasRelationType>
202                     <SPL:name rdf:datatype="http://www.w3.org↩
                           /2001/XMLSchema#string"
203                     >Problema Preencher Lacunas</SPL:name>
204                     <SPL:isChildOf rdf:resource="#Problema"/>
205                   </SPL:Feature>
206                 </SPL:isParentOf>
207                 <SPL:isParentOf>
208                   <SPL:Feature rdf:ID="↩
                         ProblemaRelacionarColunas">
209                     <SPL:isChildOf rdf:resource="#Problema"/>
210                     <SPL:hasRelationType rdf:datatype="http://↩
                           www.w3.org/2001/XMLSchema#string"
211                     >optional</SPL:hasRelationType>
212                     <SPL:name rdf:datatype="http://www.w3.org↩
                           /2001/XMLSchema#string"
213                     >Problema Relacionar Colunas</SPL:name>
214                   </SPL:Feature>
215                 </SPL:isParentOf>
216                 <SPL:isParentOf>
217                   <SPL:Feature rdf:ID="ProblemaVerdadeiroFalso"↩
                         >
218                     <SPL:hasRelationType rdf:datatype="http://↩
                           www.w3.org/2001/XMLSchema#string"
219                     >optional</SPL:hasRelationType>
220                     <SPL:name rdf:datatype="http://www.w3.org↩
                           /2001/XMLSchema#string"
221                     >Problema Verdadeiro ou Falso</SPL:name>
222                     <SPL:isChildOf rdf:resource="#Problema"/>
223                   </SPL:Feature>
```

```
224                 </SPL:isParentOf>
225                 <SPL:isParentOf>
226                   <SPL:Feature rdf:ID="ProblemaMultiplaEscolha"↩
                        >
227                     <SPL:hasRelationType rdf:datatype="http://↩
                          www.w3.org/2001/XMLSchema#string"
228                     >mandatory</SPL:hasRelationType>
229                     <SPL:isChildOf rdf:resource="#Problema"/>
230                     <SPL:name rdf:datatype="http://www.w3.org↩
                          /2001/XMLSchema#string"
231                     >Problema Multipla Escolha</SPL:name>
232                   </SPL:Feature>
233                 </SPL:isParentOf>
234               </SPL:Feature>
235             </SPL:isParentOf>
236             <SPL:isParentOf>
237               <SPL:Feature rdf:ID="Exemplo">
238                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                      XMLSchema#string"
239                 >Exemplo</SPL:name>
240                 <SPL:hasRelationType rdf:datatype="http://www.↩
                      w3.org/2001/XMLSchema#string"
241                 >optional</SPL:hasRelationType>
242                 <SPL:isChildOf rdf:resource="#Recurso"/>
243               </SPL:Feature>
244             </SPL:isParentOf>
245             <SPL:isParentOf>
246               <SPL:Feature rdf:ID="Conteudo">
247                 <SPL:isChildOf rdf:resource="#Recurso"/>
248                 <SPL:hasRelationType rdf:datatype="http://www.↩
                      w3.org/2001/XMLSchema#string"
249                 >mandatory</SPL:hasRelationType>
250                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                      XMLSchema#string"
251                 >Conteudo</SPL:name>
252               </SPL:Feature>
253             </SPL:isParentOf>
254             <SPL:isParentOf rdf:resource="#Tatica"/>
255             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                  XMLSchema#string"
256             >Recurso</SPL:name>
```

```
257                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
258                 >mandatory</SPL:hasRelationType>
259                 <SPL:isChildOf>
260                   <SPL:Feature rdf:ID="Curriculum">
261                     <SPL:isChildOf>
262                       <SPL:Feature rdf:ID="ModeloDominio">
263                         <SPL:name rdf:datatype="http://www.w3.org↩
                             /2001/XMLSchema#string"
264                         >Modelo de Dominio</SPL:name>
265                         <SPL:hasRelationType rdf:datatype="http://↩
                             www.w3.org/2001/XMLSchema#string"
266                         >mandatory</SPL:hasRelationType>
267                         <SPL:isParentOf rdf:resource="#Curriculum"/↩
                             >
268                       </SPL:Feature>
269                     </SPL:isChildOf>
270                     <SPL:hasRelationType rdf:datatype="http://www.↩
                         w3.org/2001/XMLSchema#string"
271                     >mandatory</SPL:hasRelationType>
272                     <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                         XMLSchema#string"
273                     >Curriculum</SPL:name>
274                     <SPL:isParentOf rdf:resource="#Recurso"/>
275                   </SPL:Feature>
276                 </SPL:isChildOf>
277               </SPL:Feature>
278             </SPL:isChildOf>
279             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
280             >Tatica</SPL:name>
281           </SPL:Feature>
282         </SPL:isChildOf>
283         <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
              string"
284         >Wiki</SPL:name>
285         <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
              XMLSchema#string"
286         >optional</SPL:hasRelationType>
287       </SPL:Feature>
288     </SPL:hasNodes>
289     <SPL:hasNodes>
```

```
290       <SPL:Feature rdf:ID="VisualizacaoDica">
291         <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
              XMLSchema#string"
292         >optional</SPL:hasRelationType>
293         <SPL:isChildOf>
294           <SPL:Feature rdf:ID="VisualizacaoRecurso">
295             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                  XMLSchema#string"
296             >VisualizacaoRecurso</SPL:name>
297             <SPL:isParentOf>
298               <SPL:Feature rdf:ID="VisualizacaoExemplo">
299                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                      org/2001/XMLSchema#string"
300                 >optional</SPL:hasRelationType>
301                 <SPL:isChildOf rdf:resource="#VisualizacaoRecurso"/↩
                      >
302                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                      XMLSchema#string"
303                 >VisualizacaoExemplo</SPL:name>
304               </SPL:Feature>
305             </SPL:isParentOf>
306             <SPL:isParentOf>
307               <SPL:Feature rdf:ID="VisualizacaoConteudo">
308                 <SPL:isChildOf rdf:resource="#VisualizacaoRecurso"/↩
                      >
309                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                      XMLSchema#string"
310                 >VisualizacaoConteudo</SPL:name>
311                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                      org/2001/XMLSchema#string"
312                 >mandatory</SPL:hasRelationType>
313               </SPL:Feature>
314             </SPL:isParentOf>
315             <SPL:isParentOf>
316               <SPL:Feature rdf:ID="VisualizacaoProblema">
317                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                      org/2001/XMLSchema#string"
318                 >mandatory</SPL:hasRelationType>
319                 <SPL:isChildOf rdf:resource="#VisualizacaoRecurso"/↩
                      >
320                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                      XMLSchema#string"
```

```
321                >VisualizacaoProblema</SPL:name>
322                <SPL:isParentOf>
323                  <SPL:Feature rdf:ID="↩
                        VisualizacaoRelacionarColunas">
324                    <SPL:hasRelationType rdf:datatype="http://www.↩
                          w3.org/2001/XMLSchema#string"
325                    >optional</SPL:hasRelationType>
326                    <SPL:isChildOf rdf:resource="#↩
                          VisualizacaoProblema"/>
327                    <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                          XMLSchema#string"
328                    >VisualizacaoRelacionarColunas</SPL:name>
329                  </SPL:Feature>
330                </SPL:isParentOf>
331                <SPL:isParentOf>
332                  <SPL:Feature rdf:ID="VisualizacaoVerdadeiroFalso"↩
                        >
333                    <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                          XMLSchema#string"
334                    >VisualizacaoVerdadeiroFalso</SPL:name>
335                    <SPL:hasRelationType rdf:datatype="http://www.↩
                          w3.org/2001/XMLSchema#string"
336                    >optional</SPL:hasRelationType>
337                    <SPL:isChildOf rdf:resource="#↩
                          VisualizacaoProblema"/>
338                  </SPL:Feature>
339                </SPL:isParentOf>
340                <SPL:isParentOf>
341                  <SPL:Feature rdf:ID="VisualizacaoMultiplaEscolha"↩
                        >
342                    <SPL:hasRelationType rdf:datatype="http://www.↩
                          w3.org/2001/XMLSchema#string"
343                    >mandatory</SPL:hasRelationType>
344                    <SPL:isChildOf rdf:resource="#↩
                          VisualizacaoProblema"/>
345                    <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                          XMLSchema#string"
346                    >VisualizacaoMultiplaEscolha</SPL:name>
347                  </SPL:Feature>
348                </SPL:isParentOf>
349                <SPL:isParentOf>
```

```
350                   <SPL:Feature rdf:ID="VisualizacaoPreencherLacunas↩
                         ">
351                     <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                           XMLSchema#string"
352                     >VisualizacaoPreencherLacunas</SPL:name>
353                     <SPL:isChildOf rdf:resource="#↩
                           VisualizacaoProblema"/>
354                     <SPL:hasRelationType rdf:datatype="http://www.↩
                           w3.org/2001/XMLSchema#string"
355                     >optional</SPL:hasRelationType>
356                   </SPL:Feature>
357                 </SPL:isParentOf>
358               </SPL:Feature>
359             </SPL:isParentOf>
360             <SPL:isParentOf rdf:resource="#VisualizacaoDica"/>
361             <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                   /2001/XMLSchema#string"
362             >mandatory</SPL:hasRelationType>
363           </SPL:Feature>
364         </SPL:isChildOf>
365         <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
              string"
366         >VisualizacaoDica</SPL:name>
367       </SPL:Feature>
368     </SPL:hasNodes>
369     <SPL:hasNodes rdf:resource="#VisualizacaoMultiplaEscolha"/>
370     <SPL:hasNodes rdf:resource="#ProblemaPreencherLacunas"/>
371     <SPL:hasNodes rdf:resource="#Geral"/>
372     <SPL:hasNodes rdf:resource="#Passo_a_Passo"/>
373     <SPL:hasNodes>
374       <SPL:Feature rdf:ID="Prova">
375         <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
              string"
376         >Prova</SPL:name>
377         <SPL:isChildOf>
378           <SPL:Feature rdf:ID="Somativa">
379             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
380             >Somativa</SPL:name>
381             <SPL:isParentOf>
382               <SPL:Feature rdf:ID="Formulario">
```

```
383                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
384                 >optional</SPL:hasRelationType>
385                 <SPL:isChildOf rdf:resource="#Somativa"/>
386                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
387                 >Formulario</SPL:name>
388               </SPL:Feature>
389             </SPL:isParentOf>
390             <SPL:isParentOf rdf:resource="#Prova"/>
391             <SPL:isChildOf>
392               <SPL:Feature rdf:ID="Avaliacao">
393                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
394                 >Avaliacao</SPL:name>
395                 <SPL:isParentOf rdf:resource="#Somativa"/>
396                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
397                 >mandatory</SPL:hasRelationType>
398               </SPL:Feature>
399             </SPL:isChildOf>
400             <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                    /2001/XMLSchema#string"
401             >mandatory</SPL:hasRelationType>
402           </SPL:Feature>
403         </SPL:isChildOf>
404         <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
              XMLSchema#string"
405         >mandatory</SPL:hasRelationType>
406       </SPL:Feature>
407     </SPL:hasNodes>
408     <SPL:hasNodes rdf:resource="#Avaliacao"/>
409     <SPL:hasNodes rdf:resource="#Exemplo"/>
410     <SPL:hasNodes>
411       <SPL:Feature rdf:ID="NivelConhecimento">
412         <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
              string"
413         >Nivel de Conhecimento</SPL:name>
414         <SPL:isChildOf>
415           <SPL:Feature rdf:ID="RelatorioAprendizagem">
416             <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                    /2001/XMLSchema#string"
```

```
417             >optional</SPL:hasRelationType>
418             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
419             >Relatorio de Aprendizagem</SPL:name>
420             <SPL:isParentOf>
421               <SPL:Feature rdf:ID="DesempenhoCurriculum">
422                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
423                 >Desempenho no Curriculum</SPL:name>
424                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
425                 >optional</SPL:hasRelationType>
426                 <SPL:isChildOf rdf:resource="#RelatorioAprendizagem↩
                       "/>
427               </SPL:Feature>
428             </SPL:isParentOf>
429             <SPL:isParentOf>
430               <SPL:Feature rdf:ID="HistoricoResolucoes">
431                 <SPL:isChildOf rdf:resource="#RelatorioAprendizagem↩
                       "/>
432                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
433                 >Historico de Resolucoes</SPL:name>
434                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
435                 >optional</SPL:hasRelationType>
436               </SPL:Feature>
437             </SPL:isParentOf>
438             <SPL:isParentOf>
439               <SPL:Feature rdf:ID="DesempenhoDominio">
440                 <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                       XMLSchema#string"
441                 >Desempenho no Dominio</SPL:name>
442                 <SPL:isChildOf rdf:resource="#RelatorioAprendizagem↩
                       "/>
443                 <SPL:hasRelationType rdf:datatype="http://www.w3.↩
                       org/2001/XMLSchema#string"
444                 >optional</SPL:hasRelationType>
445               </SPL:Feature>
446             </SPL:isParentOf>
447             <SPL:isParentOf rdf:resource="#NivelConhecimento"/>
448           </SPL:Feature>
```

```
449         </SPL:isChildOf>
450         <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
               XMLSchema#string"
451         >optional</SPL:hasRelationType>
452       </SPL:Feature>
453     </SPL:hasNodes>
454     <SPL:hasNodes rdf:resource="#Socratica"/>
455     <SPL:hasNodes rdf:resource="#Formulario"/>
456     <SPL:hasNodes rdf:resource="#DesempenhoDominio"/>
457     <SPL:hasNodes rdf:resource="#VisualizacaoConteudo"/>
458     <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string↩
          "
459     >Sistema Tutor Inteligente</SPL:name>
460     <SPL:hasNodes rdf:resource="#ProblemaVerdadeiroFalso"/>
461     <SPL:hasNodes>
462       <SPL:Feature rdf:ID="ModeloEstudante">
463         <SPL:isParentOf>
464           <SPL:Feature rdf:ID="Aberto">
465             <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                   /2001/XMLSchema#string"
466             >optional</SPL:hasRelationType>
467             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
468             >Aberto</SPL:name>
469             <SPL:isChildOf rdf:resource="#ModeloEstudante"/>
470           </SPL:Feature>
471         </SPL:isParentOf>
472         <SPL:hasRelationType rdf:datatype="http://www.w3.org/2001/↩
               XMLSchema#string"
473         >mandatory</SPL:hasRelationType>
474         <SPL:name rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
               string"
475         >Modelo do Estudante</SPL:name>
476         <SPL:isChildOf>
477           <SPL:Feature rdf:ID="Cadastro">
478             <SPL:hasRelationType rdf:datatype="http://www.w3.org↩
                   /2001/XMLSchema#string"
479             >mandatory</SPL:hasRelationType>
480             <SPL:isParentOf rdf:resource="#ModeloEstudante"/>
481             <SPL:name rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
482             >Cadastro</SPL:name>
```

```
483             </SPL:Feature>
484           </SPL:isChildOf>
485         </SPL:Feature>
486       </SPL:hasNodes>
487       <SPL:hasNodes rdf:resource="#HistoricoResolucoes"/>
488       <SPL:hasNodes rdf:resource="#Conteudo"/>
489       <SPL:hasNodes rdf:resource="#ExplicacaoResolucaoProblema"/>
490       <SPL:description rdf:datatype="http://www.w3.org/2001/XMLSchema↩
            #string"
491       >Uma linha de produto para STIs</SPL:description>
492       <SPL:hasNodes rdf:resource="#Problema"/>
493       <SPL:hasNodes rdf:resource="#VisualizacaoExemplo"/>
494       <SPL:hasNodes rdf:resource="#Construtivista"/>
495       <SPL:hasNodes rdf:resource="#VisualizacaoProblema"/>
496       <SPL:hasNodes rdf:resource="#Aberto"/>
497       <SPL:hasNodes rdf:resource="#Estudante"/>
498       <SPL:hasNodes rdf:resource="#VisualizacaoVerdadeiroFalso"/>
499       <SPL:hasNodes rdf:resource="#Comportamentalista"/>
500       <SPL:hasNodes rdf:resource="#Curriculum"/>
501       <SPL:hasNodes rdf:resource="#VisualizacaoPreencherLacunas"/>
502       <SPL:hasNodes rdf:resource="#VisualizacaoRecurso"/>
503       <SPL:hasNodes rdf:resource="#Tutor"/>
504       <SPL:hasNodes rdf:resource="#Cadastro"/>
505       <SPL:hasNodes rdf:resource="#Somativa"/>
506       <SPL:hasNodes rdf:resource="#Tatica"/>
507       <SPL:hasNodes rdf:resource="#↩
            EstrategiaBaseadaTeoriaAprendizagem"/>
508       <SPL:hasNodes rdf:resource="#Cognitiva"/>
509       <SPL:hasNodes rdf:resource="#Recomendar_par"/>
510       <SPL:hasNodes rdf:resource="#ModeloDominio"/>
511       <SPL:hasNodes rdf:resource="#ModeloPedagogicoBaseadoProblema"/>
512       <SPL:hasNodes rdf:resource="#Dica"/>
513       <SPL:hasNodes rdf:resource="#Recurso"/>
514       <SPL:hasNodes rdf:resource="#DesempenhoCurriculum"/>
515       <SPL:hasNodes rdf:resource="#VisualizacaoRelacionarColunas"/>
516       <SPL:hasNodes rdf:resource="#ProblemaRelacionarColunas"/>
517       <SPL:hasNodes rdf:resource="#RelatorioAprendizagem"/>
518       <SPL:hasNodes rdf:resource="#Situada"/>
519       <SPL:hasNodes rdf:resource="#ProblemaMultiplaEscolha"/>
520     </SPL:SoftwareProductLine>
521 </rdf:RDF>
```

C.4 ONTOLOGIA LEARNER

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns="http://www.nees.com.br/ontologies/sautolines/Learner.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.nees.com.br/ontologies/sautolines/Learner.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Affiliation"/>
  <owl:Class rdf:ID="ContactInfo"/>
  <owl:Class rdf:ID="Name">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    ></rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Address"/>
  <owl:Class rdf:ID="User">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    ></rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Role"/>
  <owl:Class rdf:ID="Identification"/>
  <owl:Class rdf:ID="Demographics"/>
  <owl:ObjectProperty rdf:ID="hasRole">
    <rdfs:domain rdf:resource="#User"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#Role"/>
    <owl:inverseOf>
```

```
34        <owl:ObjectProperty rdf:ID="roleOfUser"/>
35      </owl:inverseOf>
36    </owl:ObjectProperty>
37    <owl:ObjectProperty rdf:ID="identificationOfUser">
38      <rdfs:domain rdf:resource="#Identification"/>
39      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          InverseFunctionalProperty"/>
40      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
41      <rdfs:range rdf:resource="#User"/>
42      <owl:inverseOf>
43        <owl:ObjectProperty rdf:ID="hasIdentification"/>
44      </owl:inverseOf>
45    </owl:ObjectProperty>
46    <owl:ObjectProperty rdf:ID="affiliationRole">
47      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
48      <rdfs:domain rdf:resource="#Affiliation"/>
49      <rdfs:range rdf:resource="#Role"/>
50    </owl:ObjectProperty>
51    <owl:ObjectProperty rdf:ID="belongsAffiliation">
52      <rdfs:range rdf:resource="#Affiliation"/>
53      <rdfs:domain rdf:resource="#User"/>
54      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
55    </owl:ObjectProperty>
56    <owl:ObjectProperty rdf:ID="hasName">
57      <rdfs:domain rdf:resource="#Identification"/>
58      <rdfs:range rdf:resource="#Name"/>
59      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
60    </owl:ObjectProperty>
61    <owl:ObjectProperty rdf:about="#hasIdentification">
62      <rdfs:range rdf:resource="#Identification"/>
63      <owl:inverseOf rdf:resource="#identificationOfUser"/>
64      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          InverseFunctionalProperty"/>
65      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
66      <rdfs:domain rdf:resource="#User"/>
67    </owl:ObjectProperty>
68    <owl:ObjectProperty rdf:ID="hasDemographics">
```

```
69      <rdfs:domain rdf:resource="#Identification"/>
70      <rdfs:range rdf:resource="#Demographics"/>
71      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
72    </owl:ObjectProperty>
73    <owl:ObjectProperty rdf:about="#roleOfUser">
74      <rdfs:range rdf:resource="#User"/>
75      <owl:inverseOf rdf:resource="#hasRole"/>
76      <rdfs:domain rdf:resource="#Role"/>
77      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           InverseFunctionalProperty"/>
78      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
79    </owl:ObjectProperty>
80    <owl:ObjectProperty rdf:ID="hasContactInfo">
81      <rdfs:domain rdf:resource="#Identification"/>
82      <rdfs:range rdf:resource="#ContactInfo"/>
83      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
84    </owl:ObjectProperty>
85    <owl:ObjectProperty rdf:ID="hasAddress">
86      <rdfs:range rdf:resource="#Address"/>
87      <rdfs:domain rdf:resource="#Identification"/>
88    </owl:ObjectProperty>
89    <owl:DatatypeProperty rdf:ID="webSite">
90      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
91      <rdfs:domain rdf:resource="#ContactInfo"/>
92    </owl:DatatypeProperty>
93    <owl:DatatypeProperty rdf:ID="firstName">
94      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
95      <rdfs:domain rdf:resource="#Name"/>
96      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
97    </owl:DatatypeProperty>
98    <owl:DatatypeProperty rdf:ID="areaCode">
99      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
100   </owl:DatatypeProperty>
101   <owl:DatatypeProperty rdf:ID="descriptionGoal">
```

```
102     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
            string"/>
103   </owl:DatatypeProperty>
104   <owl:DatatypeProperty rdf:ID="telephone">
105     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
            string"/>
106     <rdfs:domain rdf:resource="#ContactInfo"/>
107   </owl:DatatypeProperty>
108   <owl:DatatypeProperty rdf:ID="country">
109     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
            string"/>
110     <rdfs:domain rdf:resource="#Address"/>
111     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
            FunctionalProperty"/>
112   </owl:DatatypeProperty>
113   <owl:DatatypeProperty rdf:ID="login">
114     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
            FunctionalProperty"/>
115     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
            string"/>
116     <rdfs:comment xml:lang="pt">O login eh o email.</rdfs:comment>
117     <rdfs:domain rdf:resource="#User"/>
118   </owl:DatatypeProperty>
119   <owl:DatatypeProperty rdf:ID="password">
120     <rdfs:domain rdf:resource="#User"/>
121     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
            FunctionalProperty"/>
122     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
            string"/>
123   </owl:DatatypeProperty>
124   <owl:DatatypeProperty rdf:ID="middleName">
125     <rdfs:domain rdf:resource="#Name"/>
126     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
            string"/>
127     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
            FunctionalProperty"/>
128   </owl:DatatypeProperty>
129   <owl:DatatypeProperty rdf:ID="gender">
130     <rdfs:range>
131       <owl:DataRange>
132         <owl:oneOf rdf:parseType="Resource">
```

```
133           <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema↩
                 #string"
134           >Male</rdf:first>
135           <rdf:rest rdf:parseType="Resource">
136             <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
137             >Female</rdf:first>
138             <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-↩
                   rdf-syntax-ns#nil"/>
139           </rdf:rest>
140         </owl:oneOf>
141       </owl:DataRange>
142     </rdfs:range>
143     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
144     <rdfs:domain rdf:resource="#Demographics"/>
145   </owl:DatatypeProperty>
146   <owl:DatatypeProperty rdf:ID="rememberPassDescription">
147     <rdfs:domain rdf:resource="#User"/>
148     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
149   </owl:DatatypeProperty>
150   <owl:DatatypeProperty rdf:ID="priority">
151     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
152     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
153   </owl:DatatypeProperty>
154   <owl:DatatypeProperty rdf:ID="locality">
155     <rdfs:domain rdf:resource="#Address"/>
156     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
157     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
158   </owl:DatatypeProperty>
159   <owl:DatatypeProperty rdf:ID="dateAffiliation">
160     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
161     <rdfs:domain rdf:resource="#Affiliation"/>
162     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date↩
          "/>
163   </owl:DatatypeProperty>
```

```
164   <owl:DatatypeProperty rdf:ID="email">
165     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
166     <rdfs:domain rdf:resource="#ContactInfo"/>
167   </owl:DatatypeProperty>
168   <owl:DatatypeProperty rdf:ID="city">
169     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
170     <rdfs:domain rdf:resource="#Address"/>
171     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
172   </owl:DatatypeProperty>
173   <owl:DatatypeProperty rdf:ID="registerDate">
174     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date↩
          "/>
175     <rdfs:domain rdf:resource="#User"/>
176     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
177   </owl:DatatypeProperty>
178   <owl:DatatypeProperty rdf:ID="organisation">
179     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
180     <rdfs:domain rdf:resource="#Affiliation"/>
181   </owl:DatatypeProperty>
182   <owl:DatatypeProperty rdf:ID="screenName">
183     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
184     <rdfs:domain rdf:resource="#Name"/>
185     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
186   </owl:DatatypeProperty>
187   <owl:DatatypeProperty rdf:ID="postCode">
188     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
189     <rdfs:domain rdf:resource="#Address"/>
190     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
191   </owl:DatatypeProperty>
192   <owl:DatatypeProperty rdf:ID="statusRole">
193     <rdfs:domain rdf:resource="#Role"/>
194     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
```

```
195     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
196   </owl:DatatypeProperty>
197   <owl:DatatypeProperty rdf:ID="loginTime">
198     <rdfs:domain rdf:resource="#User"/>
199     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#time↩
           "/>
200     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
201     <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#↩
           string"
202     >tempo total de todas as sessoes de login.</rdfs:comment>
203   </owl:DatatypeProperty>
204   <owl:DatatypeProperty rdf:ID="mobilePhone">
205     <rdfs:domain rdf:resource="#ContactInfo"/>
206     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
207   </owl:DatatypeProperty>
208   <owl:DatatypeProperty rdf:ID="affiliationId">
209     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
210     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
211     <rdfs:domain rdf:resource="#Affiliation"/>
212   </owl:DatatypeProperty>
213   <owl:DatatypeProperty rdf:ID="experienceLevel">
214     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
215     <rdfs:range>
216       <owl:DataRange>
217         <owl:oneOf rdf:parseType="Resource">
218           <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema↩
                #string"
219           >High</rdf:first>
220           <rdf:rest rdf:parseType="Resource">
221             <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                   XMLSchema#string"
222             >Medium</rdf:first>
223             <rdf:rest rdf:parseType="Resource">
224               <rdf:first rdf:datatype="http://www.w3.org/2001/↩
                     XMLSchema#string"
225               >Low</rdf:first>
```

```
226                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-↩
                      rdf-syntax-ns#nil"/>
227              </rdf:rest>
228            </rdf:rest>
229          </owl:oneOf>
230        </owl:DataRange>
231      </rdfs:range>
232      <rdfs:domain rdf:resource="#User"/>
233    </owl:DatatypeProperty>
234    <owl:DatatypeProperty rdf:ID="descriptionAffiliation">
235      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
236      <rdfs:domain rdf:resource="#Affiliation"/>
237    </owl:DatatypeProperty>
238    <owl:DatatypeProperty rdf:ID="descriptionRole">
239      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
240      <rdfs:subPropertyOf rdf:resource="#firstName"/>
241      <rdfs:domain rdf:resource="#Role"/>
242    </owl:DatatypeProperty>
243    <owl:DatatypeProperty rdf:ID="statepr">
244      <rdfs:domain rdf:resource="#Address"/>
245      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
246      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
247    </owl:DatatypeProperty>
248    <owl:DatatypeProperty rdf:ID="status">
249      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
250      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
          string"/>
251    </owl:DatatypeProperty>
252    <owl:DatatypeProperty rdf:ID="numberOfAccesses">
253      <rdfs:domain rdf:resource="#User"/>
254      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"↩
          />
255      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
          FunctionalProperty"/>
256    </owl:DatatypeProperty>
257    <owl:DatatypeProperty rdf:ID="streetName">
```

```
258     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
259     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
260     <rdfs:domain rdf:resource="#Address"/>
261   </owl:DatatypeProperty>
262   <owl:DatatypeProperty rdf:ID="dateOfBirth">
263     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
264     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
265     <rdfs:domain rdf:resource="#Demographics"/>
266   </owl:DatatypeProperty>
267   <owl:DatatypeProperty rdf:ID="description">
268     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
269     <rdfs:domain rdf:resource="#Demographics"/>
270   </owl:DatatypeProperty>
271   <owl:DatatypeProperty rdf:ID="rate">
272     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
273     <rdfs:domain rdf:resource="#User"/>
274     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           float"/>
275   </owl:DatatypeProperty>
276   <owl:DatatypeProperty rdf:ID="lastName">
277     <rdfs:domain rdf:resource="#Name"/>
278     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
279     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
280   </owl:DatatypeProperty>
281   <owl:DatatypeProperty rdf:ID="number">
282     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
283     <rdfs:domain rdf:resource="#Address"/>
284     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#↩
           string"/>
285   </owl:DatatypeProperty>
286   <owl:DatatypeProperty rdf:ID="date">
287     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#↩
           FunctionalProperty"/>
```

```
288      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date↩
             "/>
289    </owl:DatatypeProperty>
290 </rdf:RDF>
```