Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Linha de Produto de Software baseada na Web Semântica para Sistemas Tutores Inteligentes

Alan Pedro da Silva

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Modelos Computacionais e Cognitivos

Evandro de Barros Costa e Ig Ibert Bittencourt Santana Pinto
(Orientadores)

Campina Grande, Paraíba, Brasil

Resumo

A presente pesquisa aborda a problemática envolvida na concepção e desenvolvimento em larga escala de Sistemas Tutores Inteligentes sob a perspectiva da Web Semântica. Trata-se de um tema inerentemente interdisciplinar, envolvendo, neste trabalho, uma conjunção de aspectos relacionados a Inteligência Artificial, Web Semântica e Engenharia de Software, aplicados aos ambientes educacionais em pauta. No presente trabalho, portanto, propõe-se um estudo sobre a construção efetiva de Sistemas Tutores Inteligentes inseridos como parte de ambientes virtuais de aprendizagem para educação a distância ou presencial, tendo em consideração principalmente características de desenvolvimento baseado em reuso e em larga escala. Nesse sentido, realizou-se uma investigação sobre a adequação das abordagens existentes para este propósito, em conjunto com a definição de uma arquitetura com os componentes essenciais de Sistemas Tutores Inteligentes. Esta arquitetura foi utilizada para especificar uma linha de produto que levou em consideração (i) os diferentes perfis de usuários, (ii) o fato que estes usuários não possuem conhecimento avançado em relação a computação, (iii) a heterogeneidade dos domínios de conhecimento que este tipo de sistema irá manipular e (iv) a constante mudança e evolução do objeto que este tipo de sistema manipula: O conhecimento. Dessa forma, atendendo as quatro considerações mencionadas, foi desenvolvida e testada uma linha de produto de software baseada na web semântica. Os resultados desta abordagem mostraram-se favoráveis na experimentação realizada em dois domínios diferentes, quais sejam: Programação e Física, mostrando principalmente sua efetividade relativamente aos aspectos larga escala, adaptatividade e evolução.

Palavras chave: Sistemas Tutores Inteligentes, Inteligência Artificial, Linhas de Produto de Software, Web Semântica

Abstract

This research is related to design and building effective and large scale Intelligent Tutoring Systems under the Semantic Web perspective. It deals with clearly an interdisciplinary theme which surrounds a conjunction of aspects related to Artificial Intelligence, Semantic Web and Software Engineering, applied to those kind of educational environments. Thus, this work, proposes a study about the development of Intelligent Tutoring Systems as part of Virtual Learning Environments for presence or distance education, taking to account characteristics of such as reuse and large scale development. In this sense, an assessment of the current ITS construction approaches adequacy, along with the definition of an architecture containing the Intelligent Tutoring Systems essential components. The proposed architecture was used to specify a product line that took into account (i) the different user profiles, (ii) the non-possession of advanced computing knowledge by the users (iii) the heterogeneity of knowledge domains this kind of systems will handle and (v) the constant change and evolution of the knowledge manipulated of this kind of systems. This way, satisfying the four mentioned considerations, a software product line based on the Semantic Web was developed and tested. The experiments carried out to evaluate the proposed approach, in two different domains (Programming and Physics), demonstrate that it is effective in aspects such as large scale, adaptivity and evolution.

Keywords: Intelligent Tutoring Systems, Artificial Intelligence, Software Product Line, Semantic Web.

Agradecimentos

Agradeço primeiramente a Deus, por ter me dado força para enfrentar esta batalha. E por me colocar pessoas especiais que serviram de instrumento nesta longa caminhada.

Logo em seguida, agradeço a *Dona Alice*, minha mãe, in memoriam, porque me incentivou muito, e me deu muita força, amor e carinho em toda minha vida. E da mesma forma, ao *Seu Olival*, pelo grande empenho para me proporcionar todo o conforto possível para eu poder estudar sempre em condições especiais.

Agradeço ao Professor Evandro, que teve muita paciência para me orientar e para me incentivar, principalmente nos momentos de crise que eu passei durante o doutorado. Da mesma forma, agradeço ao Ig, que tem sido uma espécie de irmão mais velho para mim, e muito por acaso orientador. Sempre me dando um conselho aqui, um sermão ali, mas, incentivando-me muito.

Agradeço aos meus amigos Olavo, Jean, Endhe, Marlos, Heitor, Anderson, Judson, Paulo, Pryscila, Thyago, Walker e Diego Dermeval. Pois estes estiverem pessoalmente comigo nos momentos mais intensivos do doutorado.

Agradeço muito ao Patrick pela sua espiritualidade e tranqüilidade, pois, através destes atributos, este meu amigo sempre conseguia reduzir a minha ansiedade e impaciência. No mesmo sentido agradeço ao Hélio Martins, porque este meu grande amigo estava sempre disposto a me ajudar, seja lá no que fosse.

Agradeço também ao Instituto Federal de Alagoas, campus Palmeira dos Índios, e sobretudo ao Carlos Guedes e ao Emerson, porque graças ao alívio da carga horária, e de outros compromissos, foi possível dedicar mais tempo na minha tese.

Agradeço muito aos meus irmãos Marcelo, Haglay, Egmar, Edjane, Egnaldo e Egberto, pois eles foram muito importantes como retaguarda em todo o processo. Agradeço aos meus cunhados Felício, Marcos Machado e Tina, que sempre torceram muito por mim. E ao meu Tio José Nunes, que sempre vinha com uma história interessante para me incentivar no doutorado.

Conteúdo

Lista de Símbolos

PROEJA - *Programa Nacional de Integração da Educação Profissional com a Educação Básica na Modalidade de Educação Jovens e Adulto*

STI - *Sistemas Tutores Inteligentes*

SMA - *Sistemas Multiagentes*

LPS - *Linha de Produto de Software*

Lista de Figuras

Capítulo 1

Introdução

O domínio de estudo abordado na presente pesquisa é o da concepção e a produção em larga escala de Sistemas Tutores Inteligentes sob a perspectiva da Web Semântica, situando-se na linha de pesquisa Modelos Computacionais e Cognitivos da Copin[1]/UFCG. Trata-se de um tema inerentemente interdisciplinar, envolvendo, neste trabalho, uma conjunção de aspectos relacionados a Inteligência Artificial, Web Semântica e Engenharia de Software, aplicados aos ambientes educacionais em pauta.

Este capítulo apresenta uma ampla motivação para investimento em pesquisa sobre sistemas tutores inteligentes, enfatizando sua importância em diferentes contextos educacionais. Além disso, destaca-se as principais questões de pesquisa, objetivos e aspectos metodológicos da presente tese.

1.1 Motivações e Contextualização da Pesquisa

As tecnologias associadas a Internet trouxeram muitos avanços na sociedade sob vários aspectos: política, economia, cultura, e também, na educação. Isso ocorre porque a Internet/*Web* fornece mecanismos relacionados ao acesso à informação e à comunicação entre pessoas, que se tornou efetivamente viável e independente de distância e de tempo. Nesse sentido, foram essas características que fizeram com que a educação a distância, uma modalidade de ensino bastante satisfatória, atingisse milhões de pessoas em todo o mundo. No Brasil, a cada dia, esta modalidade consolida-se como uma alternativa crescente de ensino.

[1]Coordenação de Pós-Graduação em Ciência da Computação.

Por exemplo, o Brasil possuía no ano de 2008 cerca de 2,5 milhões de estudantes, sendo que destes, quase 1 milhão são oriundos de alunos matriculados em cursos regulares de ensino superior, segundo a ABED[69]. Isso representou um crescimento de 213,8% de alunos matriculados em instituições de nível superior.

Todavia, para que seja viável a implementação do ensino a distância, faz-se necessário o uso de tecnologias para sua efetivação, dentre as quais podem-se destacar as páginas na web, áudio e vídeos digitais, fórum, *chat*, e-mail, documentos de texto, entre outros. Todos estes utilizando a Internet como meio de comunicação.

Nesse contexto, surgem os ambientes virtuais de aprendizagem como um mecanismo fundamental na elaboração e gerenciamento de cursos à distância. Estes ambientes têm o propósito de oferecer um suporte na interação entre professores e alunos, dado que estes se encontram separados tanto do ponto de vista físico, quanto sob o ponto de vista temporal. Este suporte se dá através das ferramentas computacionais supracitadas que auxiliam alunos e professores em todo o processo de ensino-aprendizagem. Dessa forma, conseqüentemente, tanto os professores quanto os alunos podem ser beneficiados no sentido de obter um ensino com qualidade.

No entanto, estes ambientes ainda requerem do professor um esforço significativo para que este tipo de curso seja projetado e executado de forma adequada aos alunos[88]. Primeiro porque o professor tem que preparar conteúdos específicos para essa modalidade de ensino, tendo que utilizar tecnologias específicas, segundo porque o professor pode ser requisitado pelos seus alunos em qualquer horário com alguma mensagem de dúvida. Além disso, efetivamente, o ensino a distância ainda não oferece para os alunos um processo de ensino-aprendizado com todos os recursos de um ensino tradicional[67].

Na realidade, o professor não tem condição de atender apropriadamente, de forma individualizada, os questionamentos dos alunos, principalmente sob uma perspectiva de atendimento no tempo real. Isso se deve ao fato da existência natural de uma limitação de tempo do professor, pois as dúvidas ou questionamentos podem surgir a qualquer momento. Então, do ponto de vista tecnológico, alguns benefícios devem ser incorporados aos ambientes educacionais, para que os mesmos possam realizar um ensino com uma qualidade ainda melhor.

Estes benefícios devem ser aplicados na elaboração de conteúdos educacionais, disponibilização adequada do conteúdo aos alunos, preparação e acompanhamento de avaliações[35,

90, 60, 78, 50], conforme discutido a seguir :

- Elaboração de Conteúdos Educacionais: Ambientes virtuais de aprendizagem devem possuir ferramentas com o propósito de auxiliar o professor a elaborar conteúdos educacionais na perspectiva de ensino a distância. Esse tipo de suporte é importante porque: (1) O professor, por padrão, não tem conhecimento de tecnologias da Internet, sobretudo aquelas que exigem programação. (2) As estratégias pedagógicas para o ensino a distância diferem das estratégias do ensino tradicional. Essa diferenciação se deve ao fato de que na modalidade a distância o aluno deve ser visto como um indivíduo ativo na construção de seu próprio conhecimento, onde as estratégias estão voltadas para a auto-disciplina, auto-organização e auto-aprendizado do aluno; (3) Existe uma vasta disponibilidade de conteúdos educacionais na Internet, então faz-se necessário que esse conteúdo seja reusado na elaboração de novos cursos.

- Disponibilização de conteúdo educacional: Em paralelo, estes ambientes devem prover recursos para fornecer, de forma adequada, conteúdos educacionais aos alunos. Ou seja, o ambiente deve fornecer o conteúdo relacionado à necessidade e objetivo do aluno em tempo real e de forma personalizada.

- Construção de avaliações: As avaliações em um ambiente virtual de aprendizagem devem fornecer recursos para que o aluno possa receber um *feedback* em tempo real. Para que isso seja possível, faz-se necessário o uso de técnicas computacionais sofisticadas, pois, é bastante cansativo para o professor atender todos os alunos no momento exato que surgem suas dúvidas.

Sob esta perspectiva, a comunidade da *Inteligência Artificial e Educação* tem realizado avanços significativos na integração da ciência da computação, educação e psicologia[95]. Estes avanços têm sido desenvolvidos, sobretudo, em ambientes adaptativos e interativos, para aprendizes de diferentes perfis e diferentes domínios. Como resultado de pesquisas desta comunidade, os Sistemas Tutores Inteligentes aparecem como uma proposta promissora como suporte às atividades de professores. Isso acontece em virtude destes terem como propósito central o ensino individualizado e os processos automatizados. Em paralelo, o ensino individualizado pressupõe uma relação de um professor por aluno, com isso, um

ganho importante encontra-se no sentido de que o aluno pode tirar reais dúvidas no momento em que as mesmas surgem, contribuindo, portanto, na melhoria do seu processo de aprendizagem[14]. Além disso, um Sistema Tutor Inteligente realiza esta atividade de maneira automatizada, ou seja, este processo é executado por uma entidade de software. Isso traz os benefícios relacionados a custos, escalabilidade (permitindo que grandes quantidades de alunos sejam atendidos no mesmo momento), a redução de restrições físicas e temporais. Por fim, por padrão, os Sistemas Tutores são apropriados para um ensino baseado na resolução de problemas[95]. Isso é importante porque faz com que o aluno possa aprender de uma forma ativa, ou seja, o aluno precisa refletir constantemente sobre o domínio que o mesmo está se submetendo a aprender.

Estes sistemas possuem componentes que, de forma interligada, representam o comportamento humano inerente ao processo de ensino de algum domínio de conhecimento. Essa representação de comportamento deve ser feita de tal forma, que tanto o comportamento correto, como os incorretos devem ser representados. Sendo assim, quando um aprendiz se submete a interagir com tal tipo de sistema, o mesmo deve ser corrigido caso seja identificado algum comportamento incorreto. Esse tipo de comportamento do sistema exige que alguns componentes sejam previamente e individualmente estabelecidos, tais como: componentes que contenham a representação computacional do aprendiz; componentes que contenham a representação computacional do domínio; componentes que realizem a interação com o aprendiz; componentes que possuam a capacidade de diagnosticar o aluno em seu processo de aprendizagem, de tal forma que possa ser identificado o seu nível de conhecimento, assim como os seus problemas.

Assim, em virtude da grande potencialidade dos Sistemas Tutores Inteligentes, algumas oportunidades podem ser vislumbradas em benefício dos cursos online:

1. Redução da Sobrecarga de Trabalho do Professor: Em cursos online, existe uma natural sobrecarga do trabalho do professor. Isso ocorre pelo fato de que os horários disponíveis de estudo dos alunos são muito flexíveis. Como conseqüência, o aluno pode demandar ajuda a qualquer hora, o que implica que o professor necessita estar mais disponível aos seus alunos. Com o uso de STIs, é possível delegar aos tutores parte do processo de suporte pedagógico, ou seja, parte das dúvidas e ajudas poderiam ser respondidas de maneira automatizada, reduzindo, portanto, o seu trabalho;

2. Disponibilização de um novo tipo de Recurso Educacional: Um STI oferece uma nova forma de aprendizagem aos alunos. Portanto, a aprendizagem do aluno tende a ser potencializada. Isso ocorre em virtude do STI abordar o aluno de maneira extra e diferenciada, fazendo com que o mesmo possa interagir com o conhecimento de maneira ativa.

Todavia, o processo de construção de um Sistema Tutor Inteligente (STI) permanece uma tarefa complexa, principalmente, do ponto de vista computacional, e possui uma diversidade de técnicas computacionais e oriundas da ciência cognitiva que podem ser utilizadas. Além disso, como elas não estão relacionadas, pode se tornar complicado adequá-las a um único STI. Portanto, o custo para se desenvolver um STI ainda é alto, pois, (i) faz-se necessário profissionais que tenham conhecimento de estratégias pedagógicas, profissionais que sejam especialistas no domínio de conhecimento que o STI se propõe a ensinar, engenheiros de software, engenheiros de conhecimento e especialistas em técnicas da Inteligência Artificial. (ii) Requer um estudo prévio bastante rigoroso de como as técnicas serão utilizadas, pois, uma mesma técnica da Inteligência Artificial pode ser utilizada com mais de um propósito em um mesmo STI, assim como uma determinada funcionalidade pode ser implementada com a integração de mais de uma técnica[83]. Por exemplo, a técnica de aprendizagem de máquina pode ser utilizada tanto na implementação de um mecanismo de avaliação de conhecimento do aluno como na implementação da representação do conhecimento desse aluno[10][9][79]. Por outro lado, um mecanismo de avaliação de conhecimento do aluno pode ser implementado com integração das mesmas técnicas de aprendizagem de máquina e raciocínio baseado em regras, onde cada técnica pode oferecer uma informação diferente e complementar sobre o aluno.

Entretanto, além da problemática natural da construção de um STI em particular, existem outras barreiras que fazem com que o seu processo de desenvolvimento torne-se ainda mais custoso, principalmente se comparado a softwares mais tradicionais [75][81][39][12]. O primeiro problema é que cada STI é, muitas vezes, construído independentemente de qualquer outro, portanto, dessa forma, torna-se difícil qualquer tipo de reuso de qualquer módulo do STI, sobretudo dos componentes principais (Modelo do aprendiz, modelo do domínio, mecanismo de diagnóstico e as estratégias pedagógicas). Segundo, não existe a preocupação com a evolução desses tipos de sistema, pois, pode ser que os recursos pedagógicos ofere-

cidos ao aluno não tenham sido suficientes em relação aos seus objetivos de aprendizagem, portanto, faz-se necessário que o sistema possa se reestruturar no sentido de oferecer novos recursos, como, por exemplo, o contato com outros alunos que passaram por problemas semelhantes, ou, oferecer instrução em um domínio que seja pré-requisito. Portanto, atualmente, é uma tarefa custosa embutir Sistemas Tutores Inteligentes em cursos onlines, pelos seguintes motivos[95][68][69]:

1. Pelo fato de existir milhões de alunos, matriculados em milhares de disciplinas distribuídas em seus respectivos cursos, haveria uma demanda muito grande de diferentes tutores, em virtude da grande quantidade de currículos, para poder atender a este público. Além disso, um agravante é que uma mesma disciplina poder necessitar de mais de um tutor, e mais ainda, podem ser necessários tutores que façam um papel integrador em disciplinas que sejam relacionadas;

2. Sistemas Tutores Inteligentes são sistemas de software complexos, isso implica que sua construção necessita de profissionais bastante especializados nas áreas de Inteligência Artificial, IHC, Engenharia de Conhecimento, Pedagogia, Psicologia e Professores, o que torna a sua produção bastante onerosa e minuciosa. No entanto, os únicos responsáveis por todo o projeto e execução de um curso online, via de regra, são os professores. Isso é fator complicador para se disponibilizar recursos relacionados aos Sistemas Tutores Inteligentes, pois, os seus manuseios exigem conhecimentos especializados de computação, tal como programar, o que inviabiliza a sua distribuição a professores, de um modo geral;

3. As atuais ferramentas são complexas, exigem programação, e por si só não podem atender a essa demanda: Primeiro por causa da falta de integração dos componentes essenciais, por exemplo: (i) Quando as ferramentas disponibilizam um modelo para descrição de comportamento ou de conhecimento comportamental do aluno, elas não disponibilizam mecanismos para definição de Modelo do Estudante, Modelo de Domínio e Modelo Pedagógico; (ii) E não existem mecanismos que auxiliem o autor a utilizar ou construir Estratégias Pedagógicas que serão embutidas nos tutores;

4. O Conhecimento se modifica e evolui naturalmente e constantemente: Os componentes em um tutor são implementados de forma acoplada, portanto, não há possibi-

lidade de reutilizá-los diretamente na construção de outros tutores. Além disso, como não há uma forma padrão de como os tutores funcionam, faz-se necessário um processo de adaptação. Por outro lado, levando-se em consideração que o conhecimento em cada ciência muda ou evolui de forma bastante significativa, faz-se necessário que os componentes possam ajustar as suas funcionalidades de acordo com essas mudanças;

5. Necessidade de implementação do mecanismo de Interface: As ferramentas disponibilizam poucos mecanismos para geração da Interface, baseada na especificação de um determinado tutor. Conseqüentemente, como as interfaces são construídas para um tutor em específico, há sempre um trabalho extra para implementação da Interface;

No entanto, as tecnologias que são disponibilizadas para construção de sistemas tutores inteligentes são basicamente disponibilizadas sob as seguintes perspectivas:

1. Ferramentas de Autoria: As ferramentas de autoria são criadas para fornecer um mecanismo para construção de tutores independente de um prévio conhecimento de conceitos avançadas da computação. Isso possibilita que o autor construa o seu sistema tutor a partir do conhecimento prévio da disponibilização dos elementos essenciais e seus relacionados. Assim, faz-se necessário que o autor conheça como as estratégias pedagógicas funcionam, assim como são representados os demais componentes essenciais: modelo do domínio e modelo do aprendiz. Isso significa que é necessário entender como cada componente educacional funciona para que ele seja configurado. Essa abordagem é deficiente do ponto de vista de reuso, pois, como cada STI tem a sua configuração particular, de tal forma que seus componentes educacionais são relacionados de uma maneira específica. Isso faz com que o reuso, manutenção e evolução sejam muito comprometidos. O reuso direto se torna inviável, pois cada componente essencial tem uma configuração em cada instância. A manutenção também é uma tarefa complicada, porque a mudança em um componente pode comprometer o funcionamento dos demais componentes, e, portanto, há a necessidade de reconfiguração do sistema tutor. A evolução é algo que não é previsto em aplicações deste tipo, pois as funcionalidades do tutor estão restritas à disponibilidade de recursos oferecidas por cada ferramenta de autoria[1][2][68][3].

2. *Framework*: Os *frameworks* são ferramentas que têm um poder maior de adaptação e reuso. Isso porque estes são projetados de tal forma que o usuário pode reutilizar os componentes de um sistema tutor diretamente em outro. No entanto, faz-se necessário que o projetista do tutor saiba como os componentes existentes funcionam, isso porque o fluxo do sistema é inteiramente vinculado ao *framework*, ou seja, o usuário deste não tem controle sobre o sistema, e quais são as conseqüências que a adição de cada *hotspot* acarretará ao sistema, além disso, a sua utilização é restrita a engenheiros de software e programadores. Dessa forma, os professores, de maneira geral, não podem evoluir o STIs em contextos específicos. Além disso, um outro problema é o fato das alterações são feitas estaticamente, isto é, é difícil tornar o STI uma solução adaptativa, de acordo com diferentes perfis de estudantes. Do ponto de vista de evolução, os *frameworks* são muito restritos, isso ocorre em virtude das funcionalidades do sistema terem sido previamente estabelecidas em sua implementação[71][45][37][40].

3. Construção baseada em Ontologias e Web Semântica: As ontologias vêm sendo gradativamente embutidas no processo de construção dos Sistemas Tutores Inteligentes[50][65][12]. Esta tecnologia vêm sendo utilizada com um propósito bastante importante, que é o de permitir que todos os recursos educacionais sejam anotados semanticamente, viabilizando uma configuração automatizada destes sistemas e viabilizando um compartilhamento de conhecimento. Sob a perspectiva da Web Semântica há ainda uma importância ainda mais elevada, pois é através dela que os mais diversos tipos de recursos educacionais podem ser distribuídos e compartilhados com mais autonomia[11]. No entanto, a forma como essa automatização é implementada está vinculada tanto ao padrão de anotação fornecido por cada ambiente, como ao seu próprio fluxo. Isso faz com que o reuso, manutenção e evolução estejam limitados ao padrão de anotação estabelecido em tempo de projeto de cada STI. Além disso, a manutenção e evolução de componentes acoplados às estratégias de configuração de recursos educacionais torna-se bastante restrita, pois este recurso foi projetado para ser executado em conformidade com todos os demais componentes do sistema, o que torna uma mudança algo bastante delicado e restritivo.

Assim, as atuais abordagens utilizadas para construção de STI's ainda não se apresentam

adequadas para as necessidades de uma distribuição customizada em massa, como em um cenário de educação a distância. Ou seja, faz-se necessário distribuir tutores para os diversos currículos dos diversos cursos, onde cada tutor poderá ter características diferentes. Por esta razão, para cada versão do tutor, seria necessário um engenheiro de software ou programador para que o tutor fosse adaptado.

Portanto a produção em larga escala é comprometida pelo fato de que a construção de um sistema tutor ainda é uma tarefa onerosa do ponto de vista de tempo, e também pelo fato de que o reuso oferecido nestas abordagens ainda apresentam restrições que dificultam a personalização em larga escala. Além disso, as atuais abordagens não dão suporte à evolução automatizada. Na realidade, qualquer novo recurso que necessitar ser embutido no sistema pressupõe uma reimplementação com reflexo em grande parte do sistema para que o mesmo possa ser identificado. Isto faz com que os sistemas tutores tenham um tempo de vida útil reduzido, em virtude do conhecimento estar evoluindo constantemente.

Contudo, existe um conceito crescente em Engenharia de Software que possui algumas características que proporcionam melhorias no desenvolvimento de sistemas de uma mesma família, principalmente com relação à redução de tempo, custo e flexibilidade, é a chamada Linha de Produto de Software (LPS). Uma LPS é um conjunto intensivo de sistemas de software, que compartilham e gerenciam um conjunto de características comuns, satisfazem as necessidades de um domínio de aplicação em particular e que são desenvolvidas a partir de um conjunto comum de ativos principais e de uma forma preestabelecida [24]. Assim, esta abordagem se mostra mais eficiente quando o reuso é projetado sistematicamente, pensando em uma produção customizada de produtos oriundos de uma mesma plataforma, se comparado com outras abordagens preocupadas com reuso, tais como framework, serviços e componentes[47]. Assim, o uso de uma abordagem baseada em linhas de produto de software mostra-se promissora. Isto se deriva de algumas funcionalidades e características que esta abordagem oferece[24]:

1. Customização em Massa: Uma Linha de Produto de Software permite que a reutilização dos componentes e sua utilização na construção de uma aplicação seja feita por pessoas sem conhecimento prévio de Engenharia de Software. Isso acontece porque as decisões relacionadas às diversas possibilidades de configuração foram previamente estabelecidas no nível de requisitos. A diferença, em relação aos *frameworks*, está no

nível onde isto é feito: desde os requisitos, com refinamentos sucessivos para os demais artefatos de projeto, garantindo rastreabilidade. Com isso, o usuário da linha entende cada configuração como uma aplicação, sendo transparente questões a nível de engenharia de software relacionadas a como ela será, de fato, construída. Portanto, qualquer indivíduo que entenda o domínio de aplicação que a linha oferece, a nível de requisitos, poderá construir uma aplicação particular;

2. Produção em Larga Escala: O projeto de uma Linha de Produto pressupõe uma arquitetura bastante robusta no domínio da aplicação que a mesma está relacionada. Isso acontece porque no processo de construção da arquitetura da linha, está previsto um estudo detalhado das variabilidades, e como resultado, todos os produtos possíveis que o domínio de uma aplicação poderá oferecer já devem estar contemplados. Portanto, existe apenas uma implementação para todas as aplicações disponibilizadas pela mesma linha. Isso é uma característica muito importante, pois permite uma redução significativa nos custos de uma família de produtos.

No entanto, apesar do projeto de uma linha de produto contemplar as variabilidades referentes aos produtos que a mesma irá produzir, ainda assim faz-se necessária a presença de engenheiros de software e de conhecimento para que uma linha possa evoluir. Por outro lado, a Ideia é que seja evitada a dependência desses tipos de profissionais para evoluir tais sistemas, pois o custo pode se tornar muito alto, em um contexto de ensino a distância. Então, sob este ponto de vista, esta abordagem parece complementar as atuais abordagens para construção de Sistemas Tutores Inteligentes sob uma perspectiva da Web Semântica, pois, através dela, pode-se obter um alto nível de automatização em todo o processo de construção e reconfiguração de aplicações em Linhas de Produto[76]. Além disso, o uso de ontologias proporciona o compartilhamento de conhecimento das etapas de desenvolvimento da linha de produto[42].

A Ideia é que, a partir de uma demanda de uma aplicação em um determinado momento, seja descoberta qual é a melhor configuração da aplicação segundo os recursos disponíveis no momento corrente. Isso implica que para uma mesma demanda, requisitada em momentos diferentes, as aplicações podem ser configuradas com comportamentos diferentes. Isso acontece porque no intervalo de tempo entre as requisições, ou novos recursos podem surgir,

ou recursos podem ter sido atualizados. Portanto, essa abordagem viabiliza uma evolução dinâmica e automatizada das aplicações oferecidas por uma Linha de Produto de Software.

1.2 Questões de Pesquisa

Construir sistemas tutores inteligentes para qualquer domínio de conhecimento pode ser uma tarefa bastante especializada. Isso acontece porque cada domínio pode possuir características bastante diferentes. Por exemplo, uma simulação pode ser algo interessante para o ensino de conhecimento que exige um nível de abstração alto, tais como programação e física. Por outro lado, no ensino de português, pode ser interessante questões discursivas. No entanto, questões objetivas podem ser interessantes para todos os domínios de conhecimento. Além disso, do ponto de vista de avaliação de conhecimento, pode ser que para um mesmo currículo, os alunos sejam avaliados de forma diferente, pois, alunos que estão inseridos na modalidade PROEJA[26], por exemplo, devem ser avaliados mais pelo seu esforço e avanço, do que propriamente pelo nível de conhecimento ideal, e alunos que estão concorrendo a vestibulares concorridos devem ser avaliados de forma bastante rigorosa. Por isso, podem existir muitos detalhes no momento de se produzir tutores para um determinado propósito, e por isso, deve existir um processo customizado para construção destes sistemas, de tal forma que o mesmo possa ser adequado para cada situação que o mesmo poderá ser inserido.

Todavia, como o processo de construção destes tutores é algo caro, produzir tais tutores para cada propósito em específico seria um trabalho ainda mais caro. Por isso a busca pelo reuso sempre foi uma questão importante na construção de sistemas tutores inteligentes. Por outro lado, é comum que este reuso seja algo disponível para pessoas especializadas, sobretudo para pessoas com conhecimento avançado de computação. Nesse sentido, existem as ferramentas de autoria, que tem o propósito de fornecer um mecanismo simplificado de construção dos sistemas tutores inteligentes. Estas ferramentas tem objetivos de reduzir o esforço computacional na construção dos sistemas tutores inteligentes, mas elas ainda não foram amplamente difundidas na industria. Isso acontece porque tais ferramentas não fornecem a possibilidade de construção rápida dos sistemas tutores inteligentes, pois é necessário uma planejamento de como os tutores deverão se comportar, e muitas delas exigem um conhecimento muito especializado da inteligência artificial.

Além disso, estes sistemas têm uma característica importante, eles lidam com conhecimento. Eles lidam com conhecimento do estudante, conhecimento pedagógico e o conhecimento a ser ensinado. De forma natural, cada um deles pode ser evoluído constantemente, sendo um problema porque a vida útil destes sistemas pode se tornar curta. Levando-se em consideração que, por padrão, são necessários conhecimentos especializados de computação para se utilizar mecanismos de reuso, o mesmo acontece para viabilizar algum tipo de evolução. Diante do cenário supracitado, as questões de pesquisa embutidas nesta tese são:

Q1 Como viabilizar a construção de Sistemas Tutores Inteligentes em larga escala levando-se em consideração as suas variabilidades? Essa é uma questão complexa, porque as atuais abordagens não fornecem um mecanismo de reuso que proporcione uma construção de diferentes sistemas tutores inteligentes, com combinações de requisitos diferentes, no sentido de fornecer adaptações direcionadas aos mais variados cenários de atuação. Essas adaptações atualmente necessitam de ajustes especializados, havendo, muitas vezes, um trabalho de re-engenharia.

Q2 Como permitir que tal construção seja também adaptativa para os diferentes domínios de conhecimento? Por padrão, a construção de um determinado tutor está acoplada ao domínio de conhecimento que o mesmo está sendo projetado. Assim, se o tutor foi projetado para um determinado domínio, o mesmo terá que ser adaptado para ser utilizado em outro domínio. Levando-se em consideração a grande quantidade de domínios de conhecimento, o custo para se produzir tutores em larga escala poderia ser muito alto.

Q3 Como permitir a evolução automatizada para esse tipo de sistema? Evoluir um tutor pode ser uma tarefa bastante trabalhosa. Pois cada tutor tem as suas particularidades, e por isso, cada um deles pode estar se comportando de maneira distinta. Por isso, deve ser levado em consideração o comportamento de cada tutor, e dependendo de como os aprendizes estão se desenvolvendo, a evolução pode ser necessária de uma maneira específica para cada tutor. Por isso, evoluir de forma customizada cada tutor, que já está sendo disponibilizado e utilizado por aprendizes, pode ser uma tarefa muito trabalhosa. Então, é desejável que este trabalho possa ser realizado de maneira

automatizada.

1.3 Objetivos da Pesquisa

Após as questões de pesquisa abordadas na subseção anterior, apresentam-se aqui os objetivos de pesquisa da tese em questão.

O objetivo geral desta proposta de tese consiste na proposição de uma solução viável para a construção de sistemas tutores inteligentes em ambientes educacionais através da combinação de linhas de produto de software e a web semântica. Este objetivo pode ser remetido aos três objetivos específicos seguintes, focalizados em responder às questões anteriormente descritas.

O1. Propor uma linha de produto de software para produção de sistemas tutores inteligentes em larga escala. Desta forma, os objetivos específicos são:

- **O1.1** Fazer um estudo detalhado do domínio;
- **O1.2** Propor um modelo de *features*;
- **O1.3** Propor uma arquitetura para implementação da linha;
- **O1.4** Implementar a linha;
- **O1.5** Testar a linha.

O2. Propor uma solução computacional baseada na web semântica para construção de STIs em diferentes domínios de conhecimento. Este objetivo está relacionado a questão Q2, abordadas na subseção anterior. As questões abordam os problemas do reuso de software, no sentido de adaptação dos produtos para os diferente domínios de conhecimento. Os objetivos específicos são:

- **O2.1** Propor e/ou reusar ontologias para modelar o conhecimento que precisa ser adaptável aos tutores;
- **O2.2** Propor e/ou reusar ontologias para modelar artefatos que serão adaptáveis;
- **O2.3** Propor mecanismos de validação destas ontologias;
- **O2.4** Propor uma ferramenta para utilização destas ontologias.

O3. Propor uma solução computacional para a evolução automatizada de STIs. Este objetivo refere-se a questão Q3, e trata a questão da reconfiguração tanto da linha quanto dos produtos, quando novas *features* surgirem. Os objetivos específicos são:

O3.1 Propor e implementar uma solução para viabilizar a descoberta de novas *features*;

O3.2 Propor e implementar uma solução para atualizar a linha de produto;

O3.3 Propor e implementar uma solução para atualizar os produtos existentes.

1.4 Aspectos Metodológicos

Esta tese propõe uma nova abordagem para construção de Sistemas Tutores Inteligentes baseada em Linhas de Produto de Software e Web Semântica.

O ponto de partida para esta pesquisa de tese foi uma reflexão sobre como a construção de Sistemas Tutores Inteligentes poderia ser viabilizada em uma perspectiva de larga-escala, procurando atender necessidades tanto do desenvolvedor quanto do autor/professor. Nesse sentido, estudou-se as propostas de abordagens existentes na literatura especializada no tema em questão, optando-se pelo investimento em Linhas de Produto. Esta abordagem foi importante porque ela proporciona um reuso, também, na configuração dos produtos, proporcionando um mecanismo facilitado na construção destes, de tal forma que usuários sem conhecimentos especializados em computação possam realizar esta tarefa de uma forma autônoma. Entretanto, verificou-se logo em seguida que as abordagens tradicionais para linha de produto de software não respondiam adequadamente ao requisito de evolução. Tal constatação levou a discussão sobre a adoção de um enfoque semântico para linha de produto, o que aconteceu naturalmente em virtude da cultura de desenvolvimento de aplicações baseadas em web semântica existente no laboratório de pesquisa, do qual o autor desta tese é membro. Porém, houve antes um investimento em uma linha de produto, desenvolvendo-se, primeiramente, uma arquitetura que contemplasse requisitos importantes destes sistemas. Esta arquitetura foi baseada na experiência de vários trabalhos anteriores no âmbito do grupo de pesquisa GrOW[2], os quais, de alguma forma, podem ser resumidos em sua origem no mo-

[2]Grupo de Otimização da Web

delo Mathema[28] e mais recentemente na plataforma MASSAYO[72].

Em um segundo momento, pela necessidade de se manipular uma grande quantidade de tipos de conhecimento, que podem ser desenvolvidos por especialistas diferentes, fez-se necessário utilizar tecnologias da Web Semântica. Isto foi importante porque esta tecnologia possibilita um compartilhamento e reuso de conhecimento, de uma maneira formal e sem ambiguidades. Assim, isso possibilita que softwares possam ser construídos de uma forma mais independente do domínio de conhecimento. Portanto, baseado na premissa de se construir sistemas tutores em larga escala, utilizando-se linhas de produto de software, foi investigado como esta abordagem poderia ser utilizada, levando-se em consideração a importante característica da manipulação do conhecimento inerente ao tema central desta tese. Nesse sentido, foi constatado que as atuais abordagens de linhas de produto não foram exatamente propostas para serem utilizadas em uma perspectiva da web semântica. Por isso, foi necessário estudar meios para que fosse possível adaptar esta abordagem para este contexto.

Posteriormente, foi realizado um trabalho de descrição semântica, tanto do conhecimento que os tutores necessitam manipular, quanto ao conhecimento dos próprios artefatos da linha de produto. Em seguida, foram implementados os artefatos especificados, de tal forma que produtos poderiam ser instanciados e implementados. Portanto, os sistemas tutores foram testados em duas etapas, com alunos do Instituto Federal de Alagoas (Campus Palmeira dos Índios)[3], em domínios diferentes (Física e Programação), para analisar se os tutores produzidos realmente eram satisfatórios, e também identificar e ajustar o uso de tecnologias da web semântica nesse cenário. Com estes trabalhos, foi possível identificar a possibilidade de automatizar o processo de construção, configuração e adaptação da linha de produto e dos seus produtos aos diferentes domínios. Em paralelo, alguns resultados foram alcançados, tais como: [32][7][84][83][85].

Desse modo, verificou-se a adequação de se propor uma abordagem que combina o uso de Linhas de Produto de Software com serviços Web semânticos, visando proporcionar ambientes que garantam a construção automatizada de Sistemas Tutores Inteligente em ambientes educacionais sob uma perspectiva de larga escala, levando-se em consideração o reuso e evolução de conhecimento e software.

[3]Estes testes foram realizados com o auxílio da FAPEAL (Fundação de Amparo à Pesquisa do Estado de Alagoas), o qual aprovou um projeto de fomento de pesquisa relacionado diretamente a esta tese.

1.5 Sumário de Contribuições

Sob o ponto de vista de contribuições, foi constatado que as linhas de produto de software podem ser utilizadas para aprimorar as atuais abordagens na construção de Sistemas Tutores Inteligentes, em uma perspectiva de larga escala. Sob a perspectiva da web semântica, foi identificado que anotar semanticamente artefatos de software pode reduzir significativamente o esforço relacionado a engenharia de software, sobretudo na modelagem de aspectos cognitivos. Assim, pode-se perceber os seguintes avanços:

(i) A construção em larga escala, de forma customizada de sistemas tutores foi realizada e testada, em uma ambiente escolar. Além disso, o autor de um STI não precisa conhecer aspectos relacionados ao comportamento do sistema, portanto, apenas é necessário conhecer aspectos sobre o domínio atual e as funcionalidades desejadas. Nesse sentido, o esforço e a complexidade para a construção são reduzidos a aspectos relacionados aos requisitos desejados. As ontologias foram a principal ferramenta para isso, porque elas podem representar software e conhecimento. Ademais, as ontologias fornecem um tipo particular de reutilização de software. Isso ocorre porque os artefatos podem ser construídos para diferentes domínios de aplicação, e podem ser utilizados para várias LPS e seus respectivos produtos.

(ii) A questão da adaptação dos sistemas tutores aos diferentes domínios é uma característica da modelagem dos tutores e de artefatos anotados semanticamente, pois os mesmos fazem uso de ontologias. Através destas, foi possível separar o conhecimento do software, assim, o software pode ser implementado de forma independente do domínio de sua aplicação. Assim, é possível produzir STI para qualquer domínio com uma mesma implementação.

(iii) A evolução automatizada foi viabilizada através do uso de serviços semânticos e agentes, assim, a solução encontrada nesta tese mostra a importância destas tecnologias em sistemas tutores inteligentes. Com isso, novas *features* podem ser automaticamente adicionadas tanto aos produtos já existentes, quanto a própria linha.

1.6 Estrutura do Documento

O conteúdo seguinte deste documento consta de quatro capítulos (onde esta introdução equivale ao primeiro capítulo), os demais, correspondem os apêndices:

Capítulo 2 - Fundamentação Teórica: Este capítulo descreve os fundamentos que estão por trás da concepção desta tese;

Capítulo 3 - Trabalhos Relacionados: Este capítulo descreve os trabalhos relacionados a este tese, assim como uma discussão sobre os mesmos, no sentido de avaliar as contribuições desta tese;

Capítulo 4 - Engenharia de Domínio: Este capítulo descreve as artefatos produzidos na fase de engenharia de domínio;

Capítulo 5 - Engenharia de Aplicação: Este capítulo descreve a configuração dos produtos, assim como a evolução automatizada dos mesmos;

Capítulo 6 - Estudo de Caso: Este capítulo relata os estudos de caso para validação da viabilidade da solução proposta nesta tese;

Capítulo 7 - Conclusão e Trabalhos Futuros: Neste capítulo está descrito a conclusão e trabalhos futuros desta tese.

Capítulo 2

Fundamentação Teórica

O objetivo deste capítulo é apresentar os conceitos utilizados para elaboração deste trabalho. O capítulo divide-se em três seções: Na Seção 2.1 são levantadas questões sobre Sistemas Tutores Inteligentes, sua definição, arquitetura e características. A Seção 2.2 descreve conceitos de Linha de Produto de Software (LPS), bem como: Benefícios, variabilidade de LPS, engenharia de linha de produto de software, além de descrever dois modelos contidos na literatura para modelar a variabilidade em LPS na fase de requisitos. Por fim, a Seção 2.3 descreve o processo utilizado nesse trabalho para modelar a arquitetura proposta.

2.1 Sistemas Tutores Inteligentes

Segundo [87], um Sistema Tutor Inteligente (STI) é um programa de computador que utiliza técnicas da Inteligência Artificial (IA) para representar o conhecimento e reproduzir o comportamento do professor no processo de ensino. Assim, nesse sentido, um STI deve possuir em sua estrutura básica características que envolvam as seguintes questões: O que ensinar? Como ensinar? E a quem ensinar?

Representar o que, como e a quem ensinar requer uma fundamentação de várias disciplinas acadêmicas, incluindo ciência da computação, psicologia e educação (A Figura 2.1 ilustra o posicionamento de STIs nas disciplinas que envolvem as questões citadas acima). Estes suprem quase que completamente a cobertura da área de inteligência artificial e educação [94]. Dessa forma: (i) Inteligência artificial trata como discutir sobre inteligência e por conseguinte aprendizagem; (ii) Psicologia, particularmente a subárea de ciência cognitiva,

trata como as pessoas pensam e aprendem; e (iii) educação foca em como apoiar o ensino de uma melhor forma.

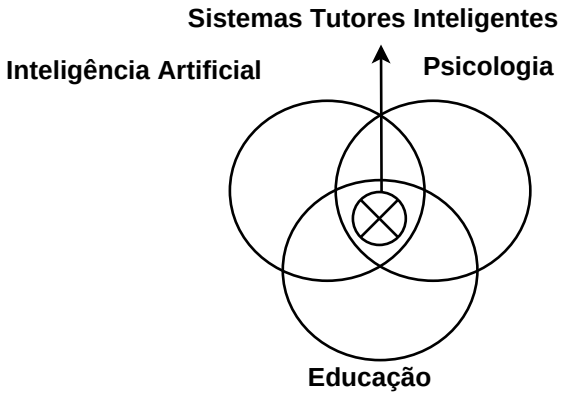


Figura 2.1: Posicionamento de STIs em três disciplinas: ciência da computação, psicologia, e educação. Adaptado de [94]

2.1.1 Descrição de um Sistema Tutor Inteligente

Do ponto de vista formal de um tutor, o primeiro componente a ser modelado é o domínio do problema, ou seja, o conhecimento relacionado ao domínio de conhecimento que o tutor se propõe a ensinar. Esse conhecimento não está relacionado apenas aos aspectos conceituais, mas, também, aos aspectos comportamentais do mesmo. Isso implica que deve-se ser modelado tanto os possíveis comportamentos que um indivíduo pode se submeter ao sistema, como as implicações de cada um desses comportamentos. Portanto, [36] define que o domínio deve ser descrito conforme a Figura 2.2, e sua formalização está descrito a seguir:

Domínio do Problema (*Problem Domain*) = $\Omega = (P, B, Solution)$, onde:

P é um Conjunto de problemas (*problems*) em Ω;

B é um Conjunto de comportamentos (*behaviours*) Ω;

Solution é a relação entre P e B. *Solution* é um subconjunto de $P \times B$

01 - Um comportamento (*behaviour* - b) é um conjunto de ações para se resolver um determinado problema (p), pertencente ao domínio geral de problemas P.

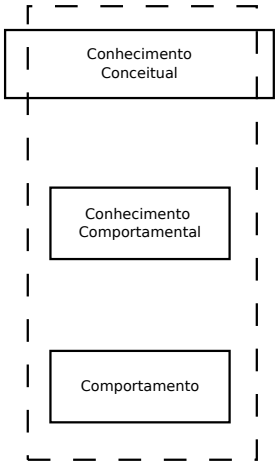


Figura 2.2: Elementos básico do conhecimento do domínio.

02 - Formalmente, esse comportamento (*behaviour*) é descrito através da seguinte sintaxe: $(agent, problem) = b_ap$. Onde, um agente *agent* tanto pode ser um aprendiz como pode ser o próprio sistema.

03 - Possíveis comportamentos: No conjunto de comportamentos é possível encontrar comportamentos corretos, incorretos e inconsistentes.

Conhecimento Comportamental (The behavioural knowledge - bk) Representa os mecanismos de inferência que avaliam o quanto um determinado comportamento é eficaz, eficiente e efetivo na resolução de um determinado problema.

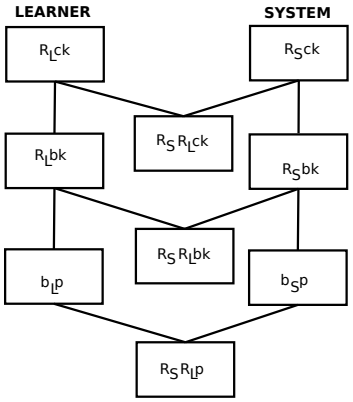
Conhecimento Conceitual (The conceptual knowledge ck) representa os conceitos e fundamentos a cerca de um determinado domínio de problema.

Perspectiva Horizontal

Sob uma perspectiva mais ampla, um Sistema Tutor Inteligente necessita representar todos os níveis de conhecimento (conhecimento comportamental, conhecimento conceitual e comportamento) tanto sobre ele mesmo, quanto a impressão dele sobre o aprendiz, conforme Figura 2.1.1.

R_Lck Representação de conhecimento conceitual (conceptual knowledge) do aprendiz;

R_Lbk Representação de conhecimento comportamental (behavioural knowledge) pelo aprendiz;



b_Lp Representação de comportamento (behaviour) para um problema p, pelo aprendiz;

R_Sck Representação de conhecimento conceitual (conceptual knowledge) do sistema;

R_Sbk Representação do conhecimento comportamental (behavioural knowledge) do sistema;

R_Sp Representação de comportamento (behaviour) para um problema p, do sistema;

R_SR_Lck Representação de conhecimento conceitual (conceptual knowledge) do aprendiz sob a perspectiva do sistema;

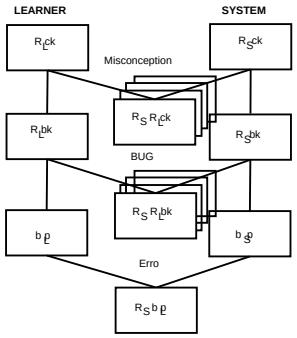
R_SR_Lbk Representação do conhecimento comportamental (behavioural knowledge) do aprendiz sob a perspectiva do sistema;

R_SR_Lp Representação de comportamento (behaviour) para um problema p, pelo aprendiz, sob a perspectiva do sistema;

No entanto, esta representação ainda não tão direta, pois, antes do tutor possuir uma imagem do aprendiz que está interagindo com ele, faz-se necessário que o tutor tenha uma imagem do que cada comportamento implica em relação a imagem do aprendiz. Nesse sentido, cada tutor necessita realizar um processo chamado de *diagnóstico*. Esse processo parte da premissa que existe uma representação completa de todos os possíveis estados de um aprendiz, em consonância com algum comportamento, conforme a Figura 2.1.1. E as fórmulas abaixo representa justamente a idéia de que o tutor tanto faz o mapeamento de comportamento correto como os incorretos.

[

$$\Psi_{bk} = \{R_SR_Lbk \mid R_SR_Lbk = R_Sbk * *(bug_1...bug_n)\}$$



$$\Psi_{ck} = \{R_S R_L ck \mid R_S R_L ck = R_S ck * *(bug_1 ... bug_n)\}$$

2.1.2 Arquitetura tradicional de um STI

O principal objetivo dos Sistemas Tutores Inteligentes é proporcionar um ensino adaptado a cada aluno, tentando se aproximar do comportamento de um professor humano na sala de aula. Contudo, para se viabilizar a sua construção, faz-se necessário que o sistema tutor tenha embutido uma grande quantidade de código relacionado a diversos aspectos bem distintos. Estes, como descrito anteriormente, buscam responder as três questões básicas: o que ensinar, como ensinar e a quem ensinar. Tais questões deságuam em modelos como descritos a seguir (Modelo do Domínio, Modelo do Aprendiz e Estratégias Pedagógicas)[94].

Modelo do Domínio - Este modelo é responsável por descrever o conteúdo que o tutor estar habilitado a ensinar, respondendo a questão o que ensinar. Ele deve ser uma representação do conhecimento especializado em um domínio específico. Dessa forma, deve representar os fatos, procedimentos ou métodos que especialistas usam para realizar tarefas ou resolver problemas.

Modelo do Aprendiz - Este modelo é responsável por modelar o comportamento e informações do estudante que vai se submeter ao processo de ensino de um tutor em algum domínio, de forma que possa haver um raciocínio automatizado em relação ao mesmo. Modelos do aprendiz tipicamente representam o comportamento do estudante, incluindo respostas do estudante, ações, resultados das ações e resultados intermediários. O comportamento do estudante reflete o conhecimento do estudante, assim como os erros mais comuns.

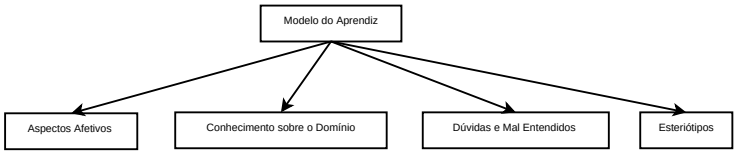


Figura 2.3: Modelo Geral do Aprendiz

Estratégias Pedagógicas - Representa as estratégias e táticas de ensino, respondendo a questão como ensinar, ele detêm o conhecimento de como vai ser ensinado, ou seja, ele elabora o sequenciamento das atividades que serão passadas para o estudante a partir de informações obtidas pelas informações modelo do aprendiz e do domínio. Conforme pode ser visto na Figura 2.4, as estratégias pedagógicas podem ser implementadas através de teorias de aprendizagem, de estratégias pedagógicas adquiridas empiricamente em experiência em situações reais ou através da elaboração de recursos computacionais que forneçam mecanismos diferentes de ensino.

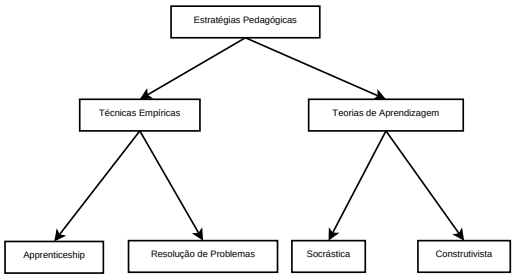


Figura 2.4: Estratégias Pedagógicas

2.2 Linha de Produto de Software

O conceito de linha de produto surgiu da necessidade de baratear a produção dos bens industriais, os quais eram feitos de forma artesanal para clientes individuais. A invenção da linha de produção possibilitou a criação dos bens para um mercado em massa a um custo menor que o do artesanal. No entanto, a linha de produção como existe, limita a diversificação dos produtos [73]. Por outro lado, a demanda por produtos individualizados é crescente, pois nem todas as pessoas desejam um mesmo tipo de produto por razões diferentes.

A linha de produto no contexto atual se baseia em dois princípios fundamentais: customização em massa e plataforma. A customização em massa se caracteriza pela produção

em larga escala de bens adaptados de acordo com as necessidades individuais do usuário [33]; já a plataforma é qualquer base de tecnologias sobre a qual outras tecnologias ou processos são construídas. A concepção de uma plataforma envolve, a priori, um mapeamento das características comuns da família de produtos para possibilitar a construção da infraestrutura que vai formar esta plataforma. O desenvolvimento de produtos baseado em plataforma em conjunto com a customização em massa permite o reuso de uma base comum de tecnologia na qual os produtos se adequam quase que totalmente aos desejos dos clientes.

Segundo [24], uma linha de produto de software (LPS) é um conjunto intensivo de sistemas de software que compartilham e gerenciam um conjunto de características comuns, satisfazem as necessidades de um segmento particular de mercado ou missão e são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma pré-estabelecida.

Da definição pode-se descrever mais detalhadamente alguns termos que representam as principais características de uma linha de produto de software:

- Conjunto comum de ativos - os ativos são a essência da linha de produto de software e correspondem a um conjunto de elementos customizáveis, utilizados na construção dos softwares produzidos (produtos). Segundo [80], eles podem ser: requisitos, análise de requisitos, modelo de domínio, arquitetura de software, engenharia de performance, documentação, planos de teste, casos de teste e dados, conhecimento humano e habilidades, processos, métodos e ferramentas, despesas, cronogramas, planos de trabalho, componentes, serviços, entre outros;

- Forma pré-estabelecida - o processo de produção de software, através de uma linha de produto, é realizado através de planos de produção. Esses planos são definidos para cada software que será produzido pela linha. Essa natureza pré-estabelecida de funcionamento do processo produtivo da linha de produto é que garante o ganho de tempo e confiabilidade no desenvolvimento dos produtos [24];

- Segmento particular de mercado - é o escopo da linha de produto e está relacionado ao domínio da família de sistemas. O escopo da linha de produto é um fator crítico para o sucesso de uma linha de produto, pois se o escopo for muito estreito (os produtos variam em um pequeno número de funcionalidades), um número insuficiente

de produtos vai ser derivado para justificar o investimento no desenvolvimento. Se o escopo for muito abrangente (produtos variam em tipo, como em funcionalidades), o esforço requerido para desenvolver produtos individuais a partir dos ativos comuns vai ser muito grande [8].

2.2.1 Benefícios

A seguir são descritas as principais motivações (benefícios) para o desenvolvimento de software sob o paradigma de engenharia de linha de produto [73]:

- Redução dos custos de desenvolvimento - O investimento inicial necessário para se construir uma linha de produto de software é maior que o da concepção de software individual, isso acontece porque o custo da criação dos ativos de software que compõem a plataforma é alto. Dada que a plataforma reusável já está criada, o custo para se desenvolver um software a partir de tal infraestrutura é reduzido, se comparado com a engenharia de software tradicional. Segundo [93], quando são construídos entre 3 e 4 sistemas a partir da linha de produto de software o investimento inicial feito equivale ao de construir entre 3 e 4 sistemas na maneira convencional, desse número em diante o custo para se construir um produto baseado na LPS é menor;

- Aumento da qualidade - Todos os artefatos criados que compõem a plataforma são revisados e testados em vários produtos diferentes. Logo, a qualidade dos produtos baseados na plataforma é aumentada através do reuso de ativos de software já testados e consolidados;

- Redução do tempo de mercado - O tempo de mercado se caracteriza pela rapidez com que um novo produto é lançado para atender uma demanda específica do mercado. Através da linha de produto de software, o tempo de mercado é diminuído, visto que vários artefatos podem ser reusados para cada novo software;

- Redução do esforço de manutenção - Quando um artefato da plataforma é mudado, para corrigir um erro por exemplo, tal mudança é propagada para todos os produtos que reusam tal artefato. Dessa forma, o esforço de manutenção é reduzido;

2.2.2 Variabilidade em Linha de Produto de Software

A noção de variabilidade contida nesta subseção é a provida por [73]. Para definir variabilidade em linha de produto faz-se uso de dois conceitos básicos: i) sujeito da variabilidade e ii) objeto da variabilidade (instância do sujeito).

O sujeito da variabilidade é um item variável do mundo real ou propriedade variável de tal item. Já o objeto da variabilidade é uma instância particular do sujeito da variabilidade. Em outras palavras, o sujeito é o que varia e o objeto é como o sujeito varia, ou qual possível forma que o sujeito pode ter. Existem diversas razões para um item ou propriedade variar: diferentes necessidades dos *stakeholders*, diferentes legislações de um país, razões técnicas, etc.

Na engenharia de linha de produto de software, os sujeitos da variabilidade e os correspondentes objetos são embutidos no contexto de linha de produto de software através das definições de ponto de variação e de variante. Um ponto de variação é uma representação de um sujeito da variabilidade dentro dos artefatos de domínio enriquecidos por informação contextual. Já a variante é uma representação de um objeto da variabilidade em artefatos de domínio.

Pontos de variação e variantes são usados para definir a variabilidade de uma linha de produto de software. [73] também propõe uma forma sistemática para identificar pontos de variação e variantes. Ela é baseada em três passos básicos:

1. Identificar o sujeito da variabilidade;
2. Identificar o ponto de variação no contexto da linha de produtos de software;
3. Definir o conjunto de variantes, onde envolve selecionar objetos da variabilidade e definí-los como variantes do ponto de variação.

A variabilidade em linha de produto de software representa os pontos de variação que serve para possibilitar o desenvolvimento de aplicações customizadas através do reuso de artefatos pré-definidos e ajustáveis. Ela envolve funcionalidades (*features*) que variam nas diferentes aplicações de uma LPS. Por outro lado, comunalidade representa funcionalidades que estão presentes em todas as aplicações baseadas da LPS.

Frisa-se ainda que existem duas visões muito importantes de variabilidade: a visão do cliente e a visão do desenvolvedor. Clientes querem aplicações customizadas para as suas necessidades individuais, isso evidencia a necessidade do cliente conhecer pelo menos uma parte da variabilidade da linha de produto de software. Por outro lado, variabilidade é parte integrante dos artefatos de domínio, logo é uma grande preocupação para quem desenvolve a LPS. Do ponto do desenvolvedor, existe um esforço maior no desenvolvimento, porque um conjunto maior de aretefatos são necessários. Por outro, comparado-se com o esforço de se desenvolver artefatos específicos para cada aplicação possível, o esforço torna-se menor. Para diferenciar essas duas percepções do que varia, existe a distinção entre variabilidade interna e externa, onde a interna representa a variabilidade que é escondida dos clientes e a externa a visível.

Essa separação entre as visões dos *stakeholders* envolvidos no desenvolvimento da LPS é importante, pois permite uma abstração dos aspectos variáveis da linha de produto, dessa forma facilita a escolha por parte do cliente das variantes que ele necessita para seu produto.

2.2.3 Documentação da Variabilidade na Fase de Requisitos

Esta subseção apresenta dois modelos utilizados neste trabalho para documentar variabilidade da linha de produto de software a nível de requisitos: a modelagem de casos de uso e a modelagem de features.

Modelagem dos Casos de Uso para Linha de Produto de Software

O modelo de casos de uso para linha de produto de software descrito nesta subseção é o proposto por [43].

Para estender a abordagem de modelagem de casos de uso para especificar os requisitos funcionais de linhas de produto de software, é necessário definir diferentes tipos de casos de uso: casos de uso *kernel* (obrigatório), que são necessárias por todos os membros da linha de produto; casos de uso opcionais, que são necessárias por alguns membros da linha de produto; e casos de uso alternativos, onde diferentes casos de uso são necessários por diferentes membros da linha de produto.

O diagrama de casos de uso [25] é a notação gráfica utilizada para representar os casos

de uso da linha de produto de software, onde os estereótipos «*kernel*», «*optional*», e «*alternative*» são usados, respectivamente, para distinguir entre os casos de uso que são sempre requeridos, casos de uso que são requiridos de vez em quando, e casos de uso na qual uma escolha deve ser feita.

A Figura 2.5 ilustra um diagrama de casos de uso onde o caso de uso *Cook Food* é *kernel* (obrigatório), e os casos de uso *Set Time of Day*, *Display Time of Day* e *Cook Food With Recipe* são opcionais.

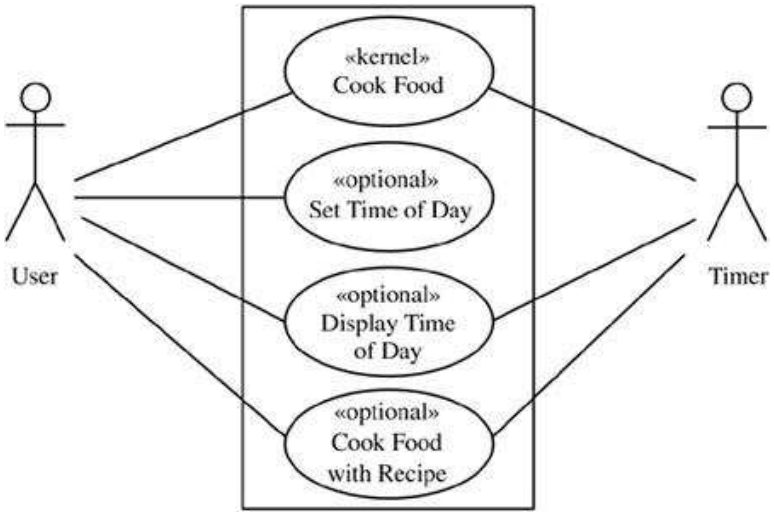


Figura 2.5: Exemplo de casos de uso *kernel* e *optional*

Da mesma forma, a Figura 2.6 ilustra um diagrama onde os casos de uso *Send Invoice* e *Bill Customer* são alternativos, ou seja, são mutuamente exclusivos.

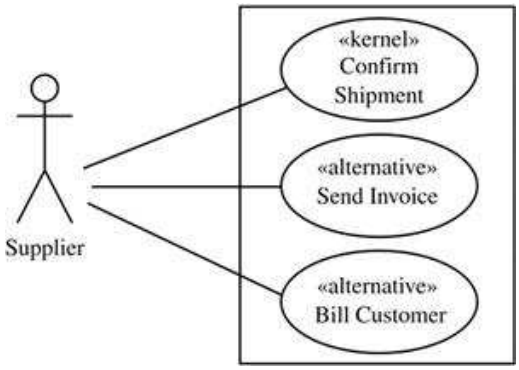


Figura 2.6: Exemplo de casos de uso *alternative* (alternativos)

Além da representação gráfica, os casos de uso da linha de produto devem ser documentados da seguinte forma:

1. **Nome do caso de uso**;
2. **Categoria de reuso**: *kernel*, *optional* ou *alternative*;
3. **Resumo**: Breve descrição do caso de uso;
4. **Atores**: Nome dos atores do caso de uso;
5. **Dependência**: Esta seção opcional descreve se o caso de uso depende de outros casos de uso, isto é, se inclui ou se estende outro caso de uso;
6. **Pré-condição**: Esta seção especifica as condições que devem ser satisfeitas para que o caso de uso seja iniciado;
7. **Descrição**: O conteúdo desta seção é uma descrição narrativa da seqüência de interações entre o ator e o sistema. O foco é nas respostas do sistema às interações, e não em como o sistema processa estas respostas;
8. **Alternativas**: Esta seção fornece uma descrição dos ramos alternativos à seqüência principal;
9. **Pontos de variação**: Esta seção define os locais na descrição do caso de uso onde funcionalidades diferentes podem ser introduzidas para membros distintos da linha de produtos. Para pequenas variações basta indicar a linha da descrição onde o caso de uso vai sofrer alteração. Já para variações complexas, onde existem grandes caminhos alternativos, os pontos de variação podem ser modelados pelos relacionamentos *include* e *extend*;
10. **Pós-condição**: Esta seção identifica a condição que sempre é verdade no final do caso de uso, isso se a sequência principal foi seguida.

Modelagem de *Features*

Uma *feature* é uma propriedade do sistema que é relevante para algum *stakeholder* e é usada para capturar semelhanças e variabilidades entre os sistemas [31]. A modelagem de *features* é uma atividade importante em linhas de produto de software, pois é nessa fase que é feita a

modelagem das funcionalidades comuns e variáveis da família de aplicações da linha de produto. As funcionalidades são organizadas no chamado modelo de *features*, que as especifica como uma árvore, onde são representadas as *features* obrigatórias, opcionais e alternativas. Essa é uma abstração interessante porque permite uma representação de fácil entendimento que pode ser usada tanto por clientes quanto por desenvolvedores. A seguir são descritos os diferentes tipos de associação entre *features*:

- Obrigatória - significa que a *feature* deve estar presente em todos os produtos construídos a partir da linha de produto de software;
- Opcional - *feature* que pode ser escolhida dependendo da aplicação, ou seja, ela pode ou não existir em um determinado produto da LPS;
- Alternativas - representa um conjunto de *features*, onde só uma pode existir em um determinado produto.
- *Or-Features* - representa um conjunto de *features*, onde um subconjunto delas pode estar presente em um produto da família de aplicações.

A notação FODA (*Feature Oriented Domain Analysis*) [51] é utilizada para representar o diagrama de *features* em forma de árvore, nessa representação é possível estabelecer os relacionamentos entre as *features*. A Figura 2.7 ilustra um exemplo de modelagem de *features* de uma loja de comércio eletrônico, além da legenda das associações das *features*. A partir da Figura 2.7 pode-se observar três *features* obrigatórias (*Catálogo*, *Pagamento* e *Segurança*); uma opcional (*Consulta*); duas *features* alternativas entre si: *Transferência Bancária* e *Cartão de Crédito*; e duas *or-features* entre si : *Alta* e *Padrão*.

Além disso, no modelo de features utilizando a notação *FODA*, é possível modelar regras de dependência entre *features*, que podem ser de dois tipos:

- *Requires* (Requer) - Esse tipo de regra significa que a escolha de uma *feature* implica na escolha de outra. Por exemplo, dadas duas *features* A e B, se A *Requires* ($\rightarrow$) B, escolher a *feature* A implica automaticamente escolher a feature B.
- *Excludes* (Exclui) - Esse tipo de regra de dependência significa que a escolha de uma *feature* implica na exclusão de outra. Por exemplo, dadas duas *features* A e B, se A *Excludes* B, a escolha da *feature* A exclui a possibilidade de escolha da *feature* B

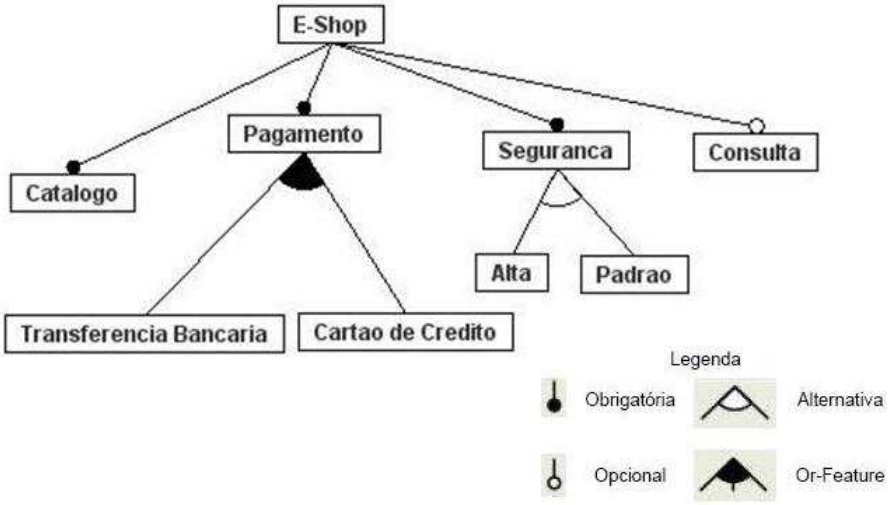


Figura 2.7: Exemplo de diagrama de *features* de uma loja de comércio eletrônico

Pode-se utilizar alguma ferramenta para criar o modelo de features, neste trabalho a ferramenta adotada foi a *pure::variants Community Edition* [74]. A Figura 2.8 explica a notação utilizada pela ferramenta na classificação de features e a Figura **??** mostra o modelo de features desenvolvido para a linha de produtos de software para lojas de comércio eletrônico descrita anteriormente.

Símbolo	Explicação
	*Feature* obrigatória
	*Feature* opcional
	*Feature* alternativa
	Indicação de que uma *feature* requer outra *feature*
	*Or-Feature*

Figura 2.8: Notação de símbolos utilizada na modelagem de *features* pela ferramenta *pure::variants*

2.2.4 Engenharia de Linha de Produto de Software

Uma outra definição muito importante no contexto de linha de produto de software é a de engenharia de linha de produto de software. Ela é dada por [73] como um paradigma para desenvolver sistemas de software usando plataformas e customização em massa. O mesmo autor separa a engenharia de linha de produto de software em dois processos:

- Engenharia de Domínio - responsável por estabelecer uma plataforma reusável e definir os aspectos comuns e variáveis da linha de produto de software para um determinado domínio, essa plataforma consiste de todos os tipos de artefatos de software (requisitos, projeto, código, testes, etc).

- Engenharia de Aplicação - responsável por derivar aplicação da linha de produto a partir da plataforma estabelecida na engenharia de domínio. Ela explora a variabilidade da linha de produto e garante a amarração correta da variabilidade de acordo com as necessidades específicas da aplicação.

A clara separação entre os processos que compõem a engenharia de linha de produto de software pode ser visto através da Figura 2.9, na qual representa o *Framework* proposto por [73]. Vale ressaltar que existem diversas metodologias para construção de linha de produto que fazem a separação entre os processos de engenharia de domínio e de aplicação, alguns são: FORM [52], KobrA [4], PLUS [43] e o Framework da Figura 2.9, que serve meramente para ilustrar tal separação, onde a camada de cima se refere à engenharia de domínio e a de baixo à de aplicação.

2.2.5 Arquitetura de Software e Arquitetura de Linha de Produto

A arquitetura de software, através de um alto nível de abstração, define o sistema em termos de seus **componentes arquiteturais**, que representam unidades abstratas do sistema; a interação entre essas entidades, que são materializadas explicitamente através dos **conectores arquiteturais**; e os atributos e funcionalidades de cada um [82]. Por conhecerem o fluxo interativo entre os componentes do sistema, é possível nos conectores, estabelecer protocolos de comunicação e coordenar a execução dos serviços que envolvam mais de um componente do sistema.

Essa visão estrutural do sistema em um alto nível de abstração proporciona benefícios importantes, que são imprescindíveis para o desenvolvimento de sistemas de software complexos. Os principais benefícios são: (i) a organização do sistema como uma composição de componentes lógicos; (ii) a antecipação da definição das estruturas de controle globais; (iii) a definição da forma de comunicação e composisão dos elementos do projeto e (iv) o auxílio na definição das funcionalidades de cada componente projetado. Além disso, uma propriedade

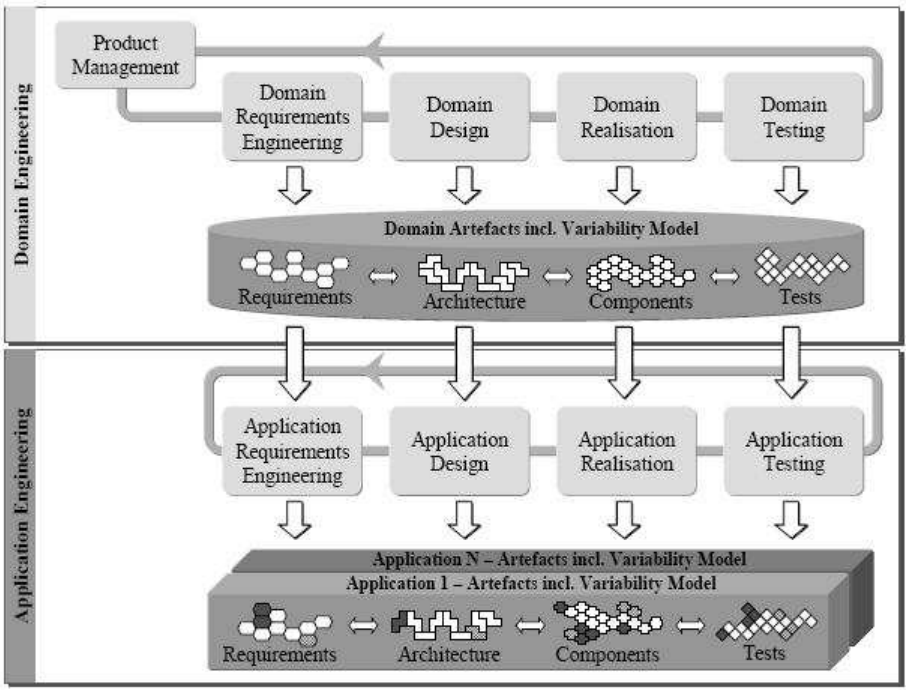


Figura 2.9: Ilustração da separação dos processos da engenharia de linha de produto de software. Adaptado de [73]

.

arquitetural representa uma decisão de projeto relacionada a algum requisito não-funcional do sistema, que quantifica determinados aspectos do seu comportamento, como confiabilidade, reusabilidade e mutabilidade [8].

A presença de uma determinada propriedade arquitetural pode ser obtida através da utilização de estilos arquiteturais que possam garantir a preservação dessa propriedade durante o desenvolvimento do sistema [82, 64]. Um **estilo arquitetural** caracteriza uma família de sistemas que são relacionados pelo compartilhamento de suas propriedades estruturais e semânticas. Esses estilos definem inclusive restrições de comunicação entre os componentes do sistema. A maneira como os componentes de um sistema ficam dispostos é conhecida como **configuração**.

As propriedades arquiteturais são derivadas dos requisitos do sistema e influenciam, direcionam e restringem todas as fases do seu ciclo de vida. Sendo assim, a arquitetura de software é um artefato essencial no processo de desenvolvimento de softwares modernos, sendo útil em todas as suas fases [21].

A importância da arquitetura fica ainda mais clara no contexto do desenvolvimento

baseado em componentes. Isso acontece, uma vez que na composição de sistemas, os componentes precisam interagir entre si para oferecer as funcionalidades desejadas e essa interação deve ser definida pela configuração arquitetural. Além disso, devido à diferença de abstração entre a arquitetura e a implementação de um sistema, um processo de desenvolvimento baseado em componentes deve apresentar uma distinção clara entre os conceitos de **componente arquitetural**, que é abstrato e não é necessariamente instanciável; e **componente de implementação**, que representa a materialização de uma especificação em alguma tecnologia específica e deve necessariamente ser instanciável.

No contexto de linhas de produto de software [46], arquitetura de linha de produto deve incorporar uma variedade de produtos similares em termos do seus elementos arquiteturais. Quanto mais variáveis estes elementos arquiteturais tenham, mais produtos diferentes podem ser gerados a partir deles. A variabilidade de software pode ser obtida adiando decisões de projeto arquitetural, e ela pode ser descrita através de pontos de variação e variantes. Portanto, um fator importante para ter sucesso na implementação de uma abordagem arquitetural de linha de produto é estruturar comunalidades e variabilidades em uma arquitetura de linha de produto em termos de componentes arquiteturais variáveis e conectores, e interfaces variáveis associadas às variantes dos pontos de variação.

2.3 *UML Components*

UML Components é um processo de desenvolvimento de software baseado em componentes. Ele é um processo simples e de fácil utilização prática, pois adota uma arquitetura prédefinida em quatros camadas [20], na qual duas destas se destacam durante o desenvolvimento: camada de sistema e camada de negócio. A camada de sistema é responsável por agrupar os componentes que implementam as funcionalidades especificadas para o sistema. Contudo, para que na prática, tais componentes possam ser implementados, eles podem necessitar de funcionalidade comuns ao domínio. Os componentes que implementam as funcionalidades estáveis do tipo de negócio ficam na camada de negócio da arquitetura. Há quatro camadas da arquitetura e o papel de cada uma pode ser vistos na Figura 2.10.

Neste processo, o desenvolvimento é dividido em seis fases, como ilustrado na Figura 2.11: i) especificação dos requisitos; ii) especificação dos componentes; iii) provisionamento

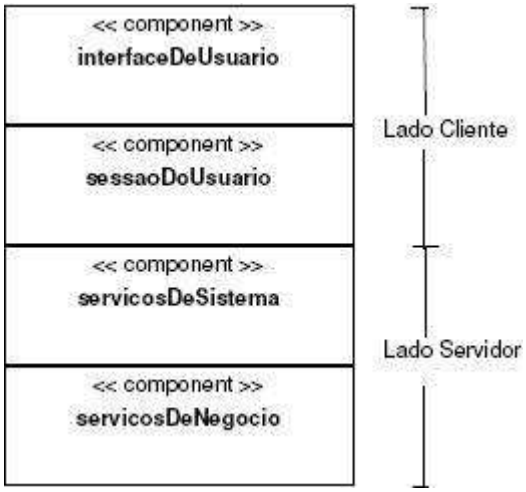


Figura 2.10: Arquitetura adotada pelo processo *UML Components*

dos componentes; iv) montagem do sistema; v) testes; e vi) implantação.

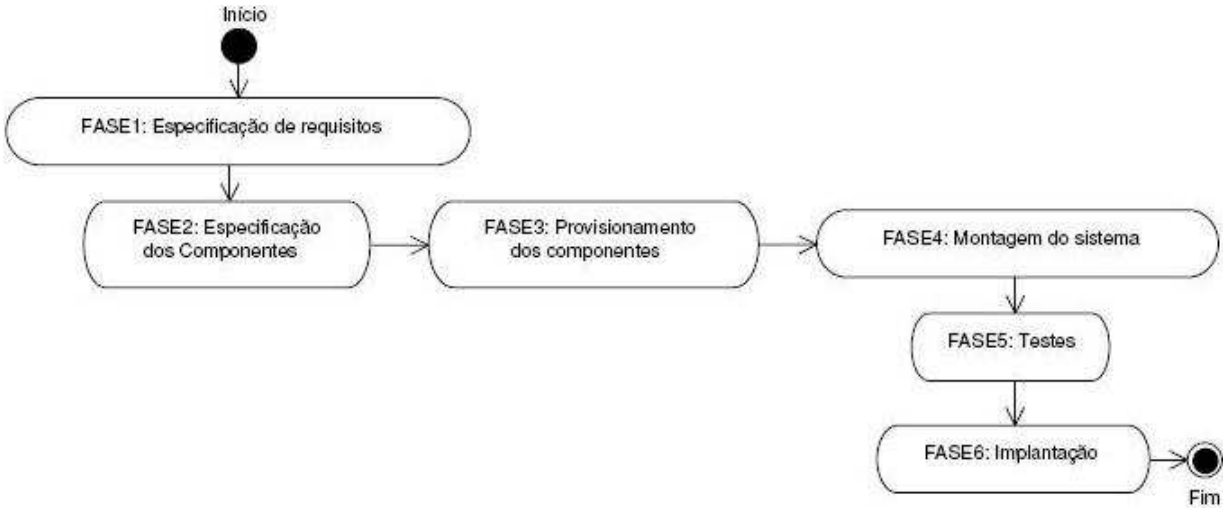


Figura 2.11: Fases do *UML Components*. Adaptado de [20]

O processo UML Components é iterativo e enfatiza com mais detalhes a fase de especificação dos componentes. Por essa razão, essa fase é dividida em três sub-fases: i) identificação dos componentes (*component identification*); ii) interação entre os componentes (*component interaction*); e iii) especificação final (*component specification*).

O escopo deste trabalho limita-se ao nível arquitetural, logo as seguintes sub-seções descrevem com mais detalhes apenas as duas fases do processo utilizadas na concepção deste trabalho: especificação dos requisitos e especificação dos componentes.

2.3.1 Especificação dos Requisitos

Os dois principais artefatos que devem ser desenvolvidos nesta fase são: diagrama de casos de uso e o modelo conceitual do negócio. Os casos de uso são utilizados para representar os requisitos funcionais do sistema, logo serão utilizados para especificar os componentes da camada de sistema. O modelo conceitual do negócio representa o domínio do problema, onde ele é a base para especificação dos componentes da camada de negócio.

2.3.2 Especificação dos Componentes

A fase de especificação dos componentes é a fase mais detalhada no processo *UML Components*. Essa fase recebe como entrada os artefatos produzidos na especificação dos requisitos.

Os principais artefatos de saída produzidos nesta fase são três: i) especificação das interfaces (refinamento das assinaturas); ii) especificação dos componentes (interfaces providas e requeridas); e iii) arquitetura do sistema. Para produzir tais artefatos, o *UML Components* divide a fase de especificação dos componentes em três estágios, a Figura 2.12 ilustra esses estágios [20]. Em seguida, faz-se uma explicação do papel de cada um desses estágios.

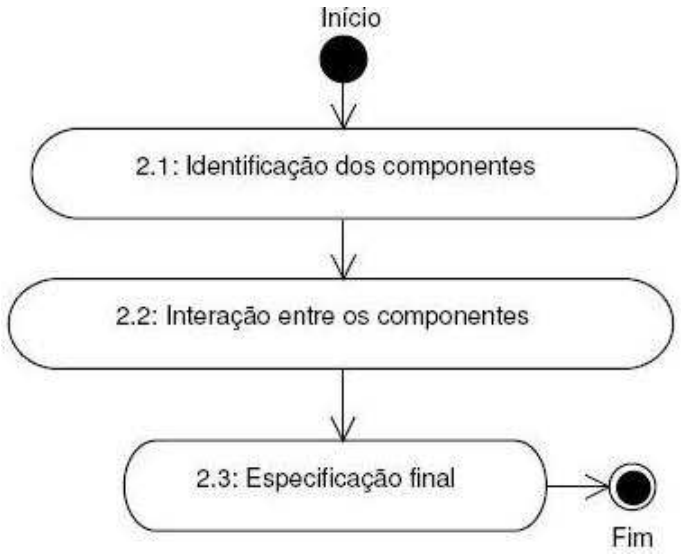


Figura 2.12: Estágios da especificação dos Componentes no *UML Components*

1. O objetivo básico do estágio de identificação dos componentes é identificar um conjunto inicial de interfaces e as primeiras operações identificadas. Além disso, essas interfaces devem ser classificadas em duas categorias: i) interfaces de sistema, relativas

às funcionalidades do sistema; e ii) interfaces de negócio, que conterão as operações básicas do negócio. As interfaces de sistema são identificadas a partir dos casos de uso e as interfaces de negócio a partir do modelo conceitual elaborado na fase de requisitos. Em seguida, para cada uma dessas interfaces, é criado um componente. Devido à classificação da interface, o componente já pode ser posicionado na arquitetura, nas camadas correspondentes à classificação da interface (sistema ou negócio).

2. Nesse estágio, o projetista examina como cada operação do sistema será implementada, ou seja, ele deverá identificar quais componentes do sistema deverão interagir para o serviço ser implementado. Para isso, o *UML Components* utiliza diagramas de colaboração UML. Através dessas colaborações, as operações requeridas do negócio são identificadas e as assinaturas das operações de sistema podem ser refinadas. No final da execução do estágio de interação entre os componentes, a arquitetura do sistema já está definida.

3. O terceiro estágio é o da especificação final dos componentes, nesta etapa as especificações são refinadas e, se possível, representadas de uma maneira formal. Durante a especificação final, a arquitetura do sistema não deve variar.

Capítulo 3

Trabalhos Relacionados

Este capítulo fornece os principais trabalhos relacionados à proposta de tese aqui descrita. De um modo geral, sob a perspectiva da web semântica, os Sistemas Tutores Inteligentes vêm sendo desenvolvidos em Ambientes Educacionais Semânticos, os quais vêm fornecendo evoluções significativas de resultados de pesquisas. A literatura tem revelado ambientes com objetivos específicos, orientados a domínio, como abordagens que usam tecnologias de Web Semântica [61][44][29] [91][19], ou através da utilização de ontologias [66][92][5]. Além disso, pode-se observar vários consórcios entre grupos de pesquisa com o objetivo de investigar ambientes com características de evolução e semântica [53][59][15]. Outro aspecto considerado pela literatura é a proposição de modelos conceituais para construir tais sistemas [34][55][62][12].

Em particular, a discussão aqui apresentada considera como relacionados a família dos ambientes que produzem Sistemas Tutores Inteligentes em larga escala com características de evolução e semântica embutida, focalizando aspectos de engenharia de software, inteligência artificial e Web Semântica. Sob a ponto de vista da engenharia de software, deseja-se investigar a família de sistemas que possibilitem algum tipo de produção de Sistemas Tutores em larga escala. Por outro lado, a inteligência artificial avalia os ambientes que utilizem agentes de software e mecanismos inteligentes que proporcionem algum tipo de evolução. Avaliam-se ainda os ambientes que garantem a automatização dos processos. Por fim, pretende-se avaliar a utilização de recursos provenientes da Web Semântica, como a utilização de ontologias, reuso de conhecimento e serviços Web semânticos. Nessa perspectiva, descreve-se aqui um apanhado de trabalhos considerados relacionados à proposta desta

tese.

As seções seguintes abordam os trabalhos relacionados. A explanação que é dada sobre estes trabalhos leva em consideração as características e as arquiteturas presentes em cada ambiente, objetivando facilitar o seu entendimento, além de suas vantagens e limitações. Após a exposição dos trabalhos relacionados, os mesmos são comparados através das características supracitadas.

3.1 Ambiente com MAS e SOA

Este trabalho foi desenvolvido na Universidade de *Athabasca*[1], no Canadá, com o objetivo de fornecer uma arquitetura para construção de sistemas que dêem suporte a usuários a realizar tarefas, educacionais, que necessitem conhecimentos especiais para a sua realização, tais como Sistemas Tutores Inteligentes.

Esta arquitetura baseia-se numa abordagem multi-agentes e arquitetura orientada a serviços [58], permitindo a construção deste tipo de sistema. A proposta possui os seguintes tipos de agentes:

- Agentes Pessoais: Estes agentes são responsáveis por gerenciar as atividades de estudantes ou professores. Assim, um usuário (estudante ou professor) pode configurar e acompanhar o seu perfil, gerenciando suas preferências. Para um determinado instrutor, este agente acompanha e gerencia o processo de criação de cursos. Por outro, para um aluno, este agente o acompanha no desenvolvimento correto de suas atividades.

- Agentes de Tarefa: Estes agentes são responsáveis no fornecimento de serviços específicos para os Agentes Pessoais, e outros agentes de tarefa, tal como os serviços de suporte a tomada de decisão. Além disso, estes agentes são responsáveis pela coordenação de serviços de outros agentes de tarefa para que possam fornecer um determinado serviço mais complexo. Por conta disto, um agente de tarefa é considerado um recurso compartilhado entre os demais usuários e outros agentes.

Além disso, esta arquitetura possui uma abordagem orientada a serviços, onde os serviços

[1] `http://www.athabascau.ca/`

são para o gerenciamento de conhecimento e recursos educacionais. A arquitetura do ambiente pode ser vista na Figura 3.1.

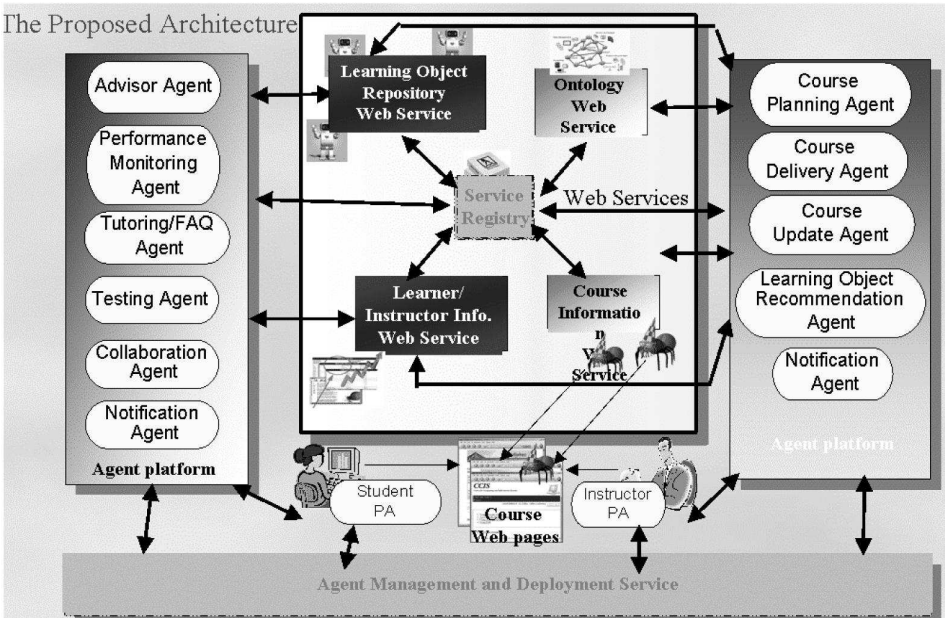


Figura 3.1: Arquitetura proposta baseada em Agentes e Serviços.

Pode-se ainda observar diversos tipos de agentes disponibilizados na plataforma de agentes. Estes agentes são utilizados para apoiar o agente pessoal do instrutor (sendo apoiado pelos agentes de planejamento de curso, distribuição do curso, atualização do curso, recomendação de objetos de aprendizagem e de notificação) e o agente pessoal do estudante (sendo apoiado pelos agentes de sugestão de objetos de aprendizagem, monitoramento de performance, tutoramento, teste, colaboração e notificação) [57].

A vantagem da arquitetura está na possibilidade de adicionar novos serviços para serem utilizados pelos agentes e na descentralização dos objetos de aprendizagem disponibilizados, entretanto, diversos problemas podem ser identificados, sobretudo em relação às questões de pesquisa:

QUESTÃO 1 Viabiliza a construção de Sistemas Tutores Inteligentes em larga escala? Este trabalho permite facilidades na construção de Sistemas Tutores Inteligentes, no entanto, estas facilidades não implicam no provimento de um mecanismo direto de configuração, como em uma Linha de Produto. Além disso, a customização dos tutores não está previamente estabelecida como em uma linha de produto, portanto, não há

uma customização em massa, principalmente para os usuários diretos dos tutores, os aprendizes;

QUESTÃO 2 Permite que tal construção seja adaptável para todos os domínios de conhecimento? Sim, isso é uma característica favorável a este trabalho, mais existem algumas ressalvas: (i) Agentes pessoais: estes agentes fazem com que haja uma limitação na arquitetura, não permitindo a adição de novos papéis para o ambiente. Isto é, caso seja necessário adicionar um monitor para utilizar o sistema, há a necessidade de criar um agente pessoal para o mesmo, além de criar uma nova plataforma com as funcionalidades específicas do monitor. (ii) Agentes de tarefas: apesar destes agentes possuírem funcionalidades fundamentais para um ambiente educacional, nenhuma facilidade é disponibilizada para a criação destes agentes. Assim, toda funcionalidade provida pelo estudante é implementada necessariamente por um único agente. Isto faz com que, para toda nova funcionalidade necessária, por mais simples que seja, necessite da criação de um novo agente com a funcionalidade, fazendo que um agente seja criado sem a real necessidade [49];

QUESTÃO 3 Permite a evolução automatizada para esse tipo de sistema? Existem algumas facilidades, por conta do sistema ser baseado em agentes. No entanto, existem limitações vinculadas as suas tecnologias: algumas limitações referentes às tecnologias são: i) Serviços Web Semânticos: a utilização de serviços Web padrões, apesar de garantir a possibilidade de adicionar novas funcionalidades nos serviços, faz com que para toda nova funcionalidade provida, haja a necessidade de manutenção nos agentes para que possa fazer a invocação destes serviços; ii) Ausência de anotação semântica: Como não há anotação semântica nos diversos recursos educacionais, não é possível prover uma evolução automatizada, pois, para que novos recursos sejam incorporados ao ambiente, uma nova implementação deve ser considerada.

3.2 ACOA

Este trabalho foi desenvolvido desenvolvido pelo grupo de pesquisa de Ciência da Computação do Instituto Politécnico Nacional[2], no México. Este trabalho aborda uma arquitetura para a construção de sistemas inteligentes tutores inteligentes (ambientes educacionais adaptativos) baseados em componentes e agentes, chamada ACOA[3].

Este ambiente proporciona o desenvolvimento de um tutor com as características essenciais destes. Além disto, o mesmo proporciona a integração de diversas tecnologias que são apropriadas para este propósito. Estas tecnologias estão dispostas nas ferramentas abaixo, que possuem as seguintes características:

1. Gerenciamento de Cursos (SiDeC): corresponde a um sistema que disponibiliza ferramentas para auxiliar a autoria de conteúdo pelos tutores;

2. Biblioteca Digital para Serviços Web (DLWS): equivale a um conjunto de recursos educacionais que provê a administração dos serviços e acesso remoto aos serviços do repositório;

3. Recursos Educacionais (IRLCOO[4]): IRLCOO é um tipo de recurso educacional sendo caracterizado por considerar características como multimídia, interatividade e feedback;

4. Avaliação dos Estudantes: este sistema é baseado na análise do perfil do estudante que é construído durante a interação com o sistema de gerenciamento de cursos.

O sistema é dividido em camadas, destacadas abaixo [30].

- Camada de Aplicação: representa a camada de comunicação entre os usuários com as ferramentas disponibilizadas no sistema. Além disso, esta camada garante a integração entre os recursos educacionais com a ferramenta de gerenciamento de cursos e a biblioteca digital para serviços Web;

[2] `http://www.ipn.mx/`

[3] Do Inglês: *Agents & Components Oriented Architecture*

[4] Do Inglês: Intelligent Reusable Learning Components Object Oriented

- Camada de Agentes e Componentes: nesta camada são disponibilizados agentes e componentes para serem utilizados pelas camadas de aplicação. Com isso, os componentes são utilizados pelo SiDeC e pela DLWS, enquanto os agentes são utilizados pelo sistema de avaliação, através da análise do perfil de estudantes construído ao longo das interações com o sistema;

As camadas descritas acima podem ser vistas na Figura 3.2.

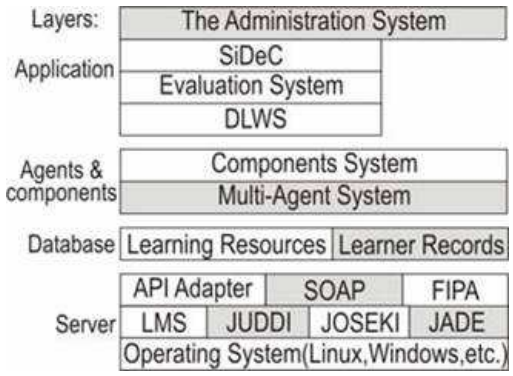


Figura 3.2: Arquitetura multicamadas proposta baseada em Agentes e Componentes.

Nesse sentido, segue a avaliação desta arquitetura:

QUESTÃO 1 Viabiliza a construção de Sistemas Tutores Inteligentes em larga escala? Sim, mas não de maneira direta. Faz-se necessário um trabalho de implementação para cada tipo de tutor; Um dos problemas foi que as tecnologias baseadas na Web Semântica foram utilizadas apenas para a avaliação de estudantes. Ou seja, tais características são abordadas no lado servidor, pela ferramenta JOSEKI[5].

QUESTÃO 2 Permite que tal construção seja adaptável para todos os domínios de conhecimento? Sim, isso está previsto no trabalho. A vantagem da proposta está no fato de utilizarem uma estrutura baseada em agentes para o sistema de avaliação. Entretanto, esta arquitetura não faz uso dos recursos provenientes da Web Semântica, buscando garantir maior adaptatividade e inteligência para o ambiente. Além disso, o ambiente não disponibiliza mecanismos inteligentes para prover mais facilidade na adaptação e a abordagem de agentes é utilizada exclusivamente para o sistema de avaliação. As-

[5] `http://www.joseki.org/`

sim, cada ambiente possui os mesmos recursos para todos os aprendizes, e portanto, não há possibilidade de se prover uma customização em massa.

QUESTÃO 3 Permite a evolução automatizada para esse tipo de sistema? Sim. No entanto, é um processo manual.

3.3 Semantic e-Learning Framework

Este trabalho foi desenvolvido pelo Centro de Computação na Internet, da Universidade de Hull[6], no Reino Unido. O mesmo propõe um framework para ambientes de *e-Learning* semânticos e inteligentes. Este *framework* leva em consideração aspectos de escalabilidade e extensibilidade como um dos seus principais requisitos.

O framework utiliza tecnologias da Web Semântica para a integração de conteúdos providos, modelo do estudante, entre outros [48]. Além da utilização de tecnologias provenientes da Web Semântica, o framework faz uso de agentes pessoais com o objetivo de prover informações para os usuários através da distribuição de serviços para aprendizagem personalizada baseado em teorias de aprendizagem. Além disso, a informação provida é baseada no perfil dos usuários. A Figura 3.3 ilustra a arquitetura do framework.

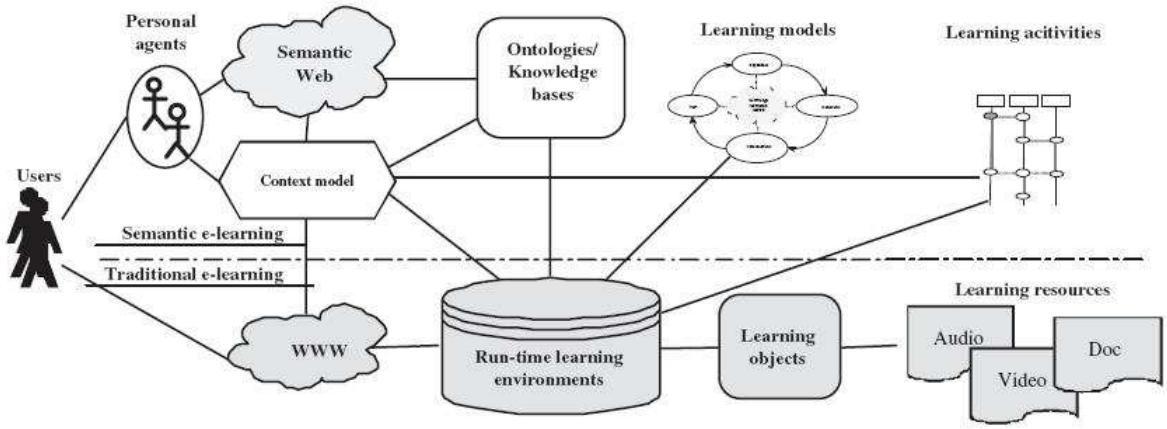


Figura 3.3: Arquitetura de um Framework para ambientes de *e-learning* semânticos e inteligentes.

Observa-se na arquitetura que duas perspectivas são mostradas, uma descrevendo uma

[6] www.hull.ac.uk/

abordagem de ambientes de *e-learning* tradicionais e outra semântica. Na perspectiva tradicional, há o ambiente educacional e os objetos de aprendizagem disponibilizados na Web. Já na perspectiva da Web semântica, há os agentes pessoais, além de bases de conhecimento/ontologias, modelos de aprendizagem e atividades de aprendizagem. Além disso, os agentes e ontologias fazem uso dos recursos tradicionais oferecidos pelos ambientes de *e-Learning*.

Apesar da arquitetura abordar utilização de tecnologias da Web Semântica nos ambientes de *e-Learning*, o mesmo não faz uso de serviços Web Semânticos, consequentemente, não garante a descoberta, composição e invocação de serviços, e também evolução. Além de não fornecer uma customização em massa.

QUESTÃO 1 Viabiliza a construção de Sistemas Tutores Inteligentes em larga escala? Sim, mas os produtos precisam ser configurados. Além disso, não está previsto a individualização dos produtos direta para os aprendizes;

QUESTÃO 2 Permite que tal construção seja adaptável para todos os domínios de conhecimento? Sim, isto é possível.

QUESTÃO 3 Permite a evolução automatizada para esse tipo de sistema? Não está previsto.

3.4 Multitutor

Este trabalho foi desenvolvido pela Escola de Administração de Negócios[7], da Universidade de Belgrado[8], na Sérvia.

A pesquisa desenvolvida pela universidade objetivou abordar a integração das tecnologias de ambientes educacionais tradicionais com sistemas tutores inteligentes e propõe os sistemas de gerenciamento de aprendizagem inteligentes. Como resultado, o sistema construído é o Multitutor, que permite que professores construam sistemas tutores para cada tipo do curso[86]. O ambiente possui os módulos tradicionais presentes em um sistema tutor inteligente, adicionando o módulo de autoria para professores especificarem características particulares de cursos. Além disso, o tutor possui um módulo especialista tendo regras como base de conhecimento. A Figura 3.4 aborda a arquitetura do Multitutor.

[7]Do Inglês: *FON - School of Business Administration*.

[8]`www.bg.ac.yu/en_index.php`

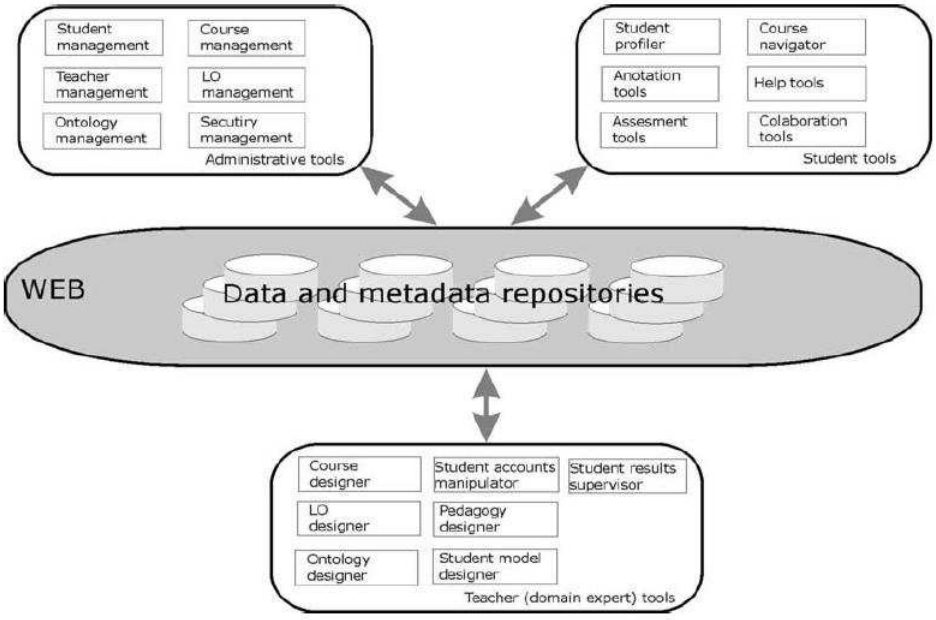


Figura 3.4: Arquitetura do Multitutor.

Apesar de prover o módulo de autoria para adicionar novos tutores baseados em cursos, o mesmo não é um framework, não permitindo que novas funcionalidades sejam adicionadas ao tutor, nem mesmo mudar o mecanismo de inferência utilizado pelo mesmo. Por isso, esta ferramenta não fornece nenhum mecanismo que viabilize a distribuição em massa de sistemas tutores.

QUESTÃO 1 Viabiliza a construção de Sistemas Tutores Inteligentes em larga escala? Sim, mas os produtos precisam ser configurados pelos professores. Por outro lado, não é possível distribuir estes sistemas diretamente para os alunos. Além disso, não está previsto a individualização dos produtos;

QUESTÃO 2 Permite que tal construção seja adaptável para todos os domínios de conhecimento? Sim, isto é possível.

QUESTÃO 3 Permite a evolução automatizada para esse tipo de sistema? Não está previsto. Até porque não é possível expandir o ambiente.

3.5 Linhas de Produto de Software Semântica

O propósito desta abordagem[63] é propor uma metodologia para desenvolvimento de linhas de produto orientada a serviços baseada em uma especialização multi-estágios de modelos de features. Nesse sentido, a Figura 3.5 ilustra a visão geral do processo global de desenvolvimento desta abordagem. A semântica nas duas primeiras fases propõe um aprimoramento tanto na análise quanto no projeto de domínio orientado a *features*, da seguinte forma:

1. **Adaptação ao contexto de orientação a serviços:** Esta etapa também sugere a especificação dos Requisitos Não-Funcionais (RNFs) relacionados com a plataforma que uma determinada aplicação será embutida. Uma vez feito isto, o próximo passo é oferecer uma especialização do modelo de features baseado em plataformas de *deployment* e com base nas preferências do usuário (ou seja, serviço de adaptação na Figura 3.5).

2. **Processo Geral:** No entanto, este trabalho[63] concentra-se apenas na primeira etapa, onde são considerados as *features* da plataforma de *deployment*. No processo geral, propõe-se o uso da web semântica em termos de regras (SWRL) e ontologias (OWL) para automatizar o processo de especialização e obter um conjunto de especializações de modelos de features para um determinado conjunto de requisitos (isto é, configuração de produtos baseada em serviços).

3. **Processo de Seleção:** O processo de seleção de *features* adequadas para uma configuração de produtos orientada a serviços é influenciada pela especificação de requisitos, que abrange requisitos para a configuração do produto. Em particular, desde a definição do RNFs rígidos, relacionados a plataforma. O RFRs são incorporados a especificação de requisitos que neste caso são servidos pelas limitações das plataformas, tais como dispositivos móveis. Por exemplo, RNFs podem descrever as características das features do aparelho para streaming de vídeo e processamento multimídia na plataforma de implantação de destino. Assim, nesta abordagem, RNFs são adicionados a uma base de conhecimento OWL como instâncias da ontologia.

4. **Processo de Configuração:** Além disso, instâncias da ontologia devem refletir todos os RNFs como restrições rígidos que excluem e incluem a seleção de serviços em modelos de features anotados semanticamente. Assim, no processo de configuração,

um subconjunto de serviços é selecionado com base em restrições rígidas especificadas pelas capacidades individuais de cada dispositivo. No processo de configuração, o modelo do dispositivo deve estar de acordo com o modelo de feature anotado. Os recursos que não preenchem os RNFs de dispositivo de destino são descartados, fazendo uma "poda" no modelo de features inicial para um modelo de novo recurso, cujo conjunto de configurações possíveis é um subconjunto do modelo de característica original. Este modelo de feature derivado é referenciado como uma especialização do modelo de *features* e o processo de refinamento. Em termos de raciocínio baseado em OWL, o objetivo é determinar se as instâncias da ontologia RNFs são consistentes com relação às propriedades de anotação definido no modelo de *features*.

5. **Validação do Modelo:** Assim, se o resultado for uma ontologia inconsistentes (ou seja, RNFs filtrados alguns recursos obrigatórios), é necessário voltar à fase de análise de recursos e de *design* para refinar a família de composições para satisfazer as inconsistências descobertas. Caso contrário, se o resultado dessa etapa é uma especialização do modelo de *features* consistentes, um novo processo de uma maior especialização e descoberta de serviços. A especialização do serviço pode ser baseada em requisitos dos usuários (por exemplo, preferências para determinados recursos) dos interessados.

Uma vez que a configuração final é obtida, ou seja, toda a variabilidade estiver sido resolvida, inicia-se o processo para a geração final e implantação do sistema orientado a serviços. Para isto, este trabalho utilizou o *Web Service Modeling Ontology*(WSMO)[16], conforme Figura 3.6. Mais especificamente, para a configuração dos recursos obtidos, é gerada uma descrição completa das composições de serviço WSMO. Nesta transformação, são gerados todos os elementos da especificação WSMO, incluindo: (i) capacidades, (ii) pré e pós-condições de transição, (iii) coreografias (juntamente com as regras de assinatura de estado e transição) e (iv) orquestrações. Nesse sentido, durante o projeto de transformações entre os serviços e modelos de *features*, para ser capaz de gerar composições serviço completo, é necessário informações sobre as propriedades não-funcionais e informações sobre a ontologia de cada *feature*. Portanto, o processo de anotações de *features*, apresentado em outro trabalho [77] complementa o processo de geração de descrições de serviço.

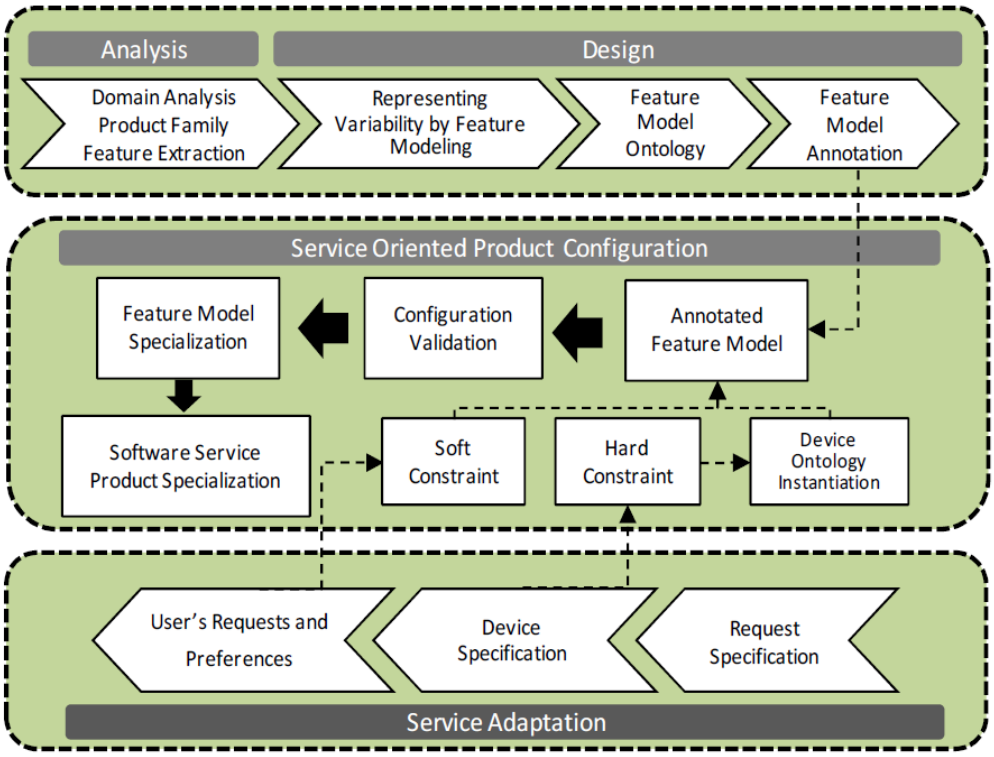


Figura 3.5: Processo mais abstrato para Linhas de Produto Semântica.

Neste trabalho[77], como mostra a Figura 3.6, o estabelecimento de uma relação entre ontologias e o processo de desenvolvimento de software é apontado como uma das conseqüências da formalização de um quadro multi-estágio com a ontologia como referência semântica. Neste contexto, uma preocupação central é garantir a rastreabilidade de artefatos e conceitos no domínio de aplicativos de negócios aos níveis de mapeamento de abstração. Alguns *plug-ins* para ferramentas de desenvolvimento de software são utilizados um protótipo para avaliar a viabilidade da adoção de uma mesma ontologia para apoiar a representação do conhecimento do negócio e para a execução de validação final de artefatos. Estas ontologias funcionam como um guia contendo os conceitos fundamentais de negócios e desenvolvimento exigidos pelas ferramentas dirigidas a modelos (*model driven*) para gerar negócios especializados e artefatos de software validado. Os artefatos são rastreados ao longo do ciclo de desenvolvimento, dos bens essenciais, através da instalação e configuração do produto final no cliente. Tendo em conta que o processo de desenvolvimento de software apoiou-se de uma linha de produtos, bens reutilizáveis e categorização são garantidos pelas ligações estabelecidas entre os artefatos e da ontologia.

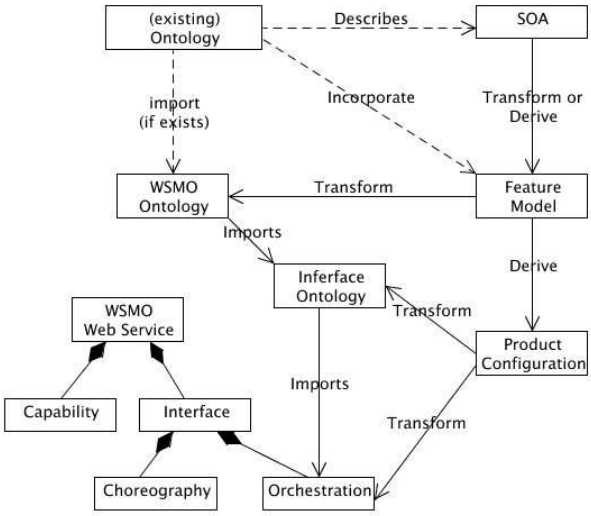


Figura 3.6: Processo mais especializado para Linhas de Produto Semântica.

QUESTÃO 1 Viabiliza a construção de Sistemas Tutores Inteligentes em larga escala? Esta abordagem não está adaptada para construção de sistemas cognitivos, tais como Sistemas Tutores Inteligentes;

QUESTÃO 2 Permite que tal construção seja adaptável para todos os domínios de conhecimento? Não está previsto a modelagem de conhecimento cognitivo. Isso acontece porque os artefatos relacionados aos requisitos não funcionais estarem expressos em um formalismo não expansivel;

QUESTÃO 3 Permite a evolução automatizada para esse tipo de sistema? Não está previsto.

3.6 Tabela Comparativa

Esta seção aborda um estudo comparativo entre as propostas citadas anteriormente. O estudo comparativo está separado na avaliação dos aspectos relacionados as questões de pesquisa desta tese, conforme a seguir:

QUESTÃO 1 Viabiliza a construção de Sistemas Tutores Inteligentes em larga escala?

QUESTÃO 2 Permite que tal construção seja adaptável para todos os domínios de conhecimento?

Tabela 3.1: Tabela Comparativa dos Trabalhos Relacionados - Questões de Pesquisa

Ambientes			
-	QUESTÃO 1	QUESTÃO 2	QUESTÃO 3
Ambiente com MAS e SOA	com restrições	sim	com restrições
ACOA	com restrições	sim	com restrições
Sem. e-Learning Fram.	com restrições	sim	não
Multitutor	com restrições	sim	não
Linhas de Produto Semântica	com restrições	com restrições	com restrições

QUESTÃO 3 Permite a evolução automatizada para esse tipo de sistema?

Assim, a tabela 3.1 descreve uma comparação dos trabalhos relacionados através da utilização das tecnologias provenientes da Web Semântica (como Ontologias e Serviços Web Semânticos) e Engenharia de Software. Assim, o resultado da comparação entre os trabalhos relacionados segue abaixo:

Capítulo 4

Engenharia de Domínio

A Engenharia de Domínio de uma linha de produto de software, como descrita na Seção 2.2, é responsável por estabelecer a plataforma reusável da linha de produto, nesta plataforma devem estar presentes todos os artefatos de software que definem os aspectos comuns e variáveis da linha de produto de software. O presente trabalho especifica os artefatos de software presentes nas fases de requisitos e projeto. Os artefatos de requisitos da plataforma estão documentados, como mostrado neste capítulo, em diagrama de casos de uso com variabilidade e diagrama de *features* (além dos diagramas que ajudam a entender o domínio de STI), o de atividades e o modelo conceitual do negócio. O processo seguido para especificar a arquitetura proposta é uma adaptação do *UML Components* [20], pois neste capítulo o foco é especificar a arquitetura da linha de produto de software. Dessa forma existem algumas estratégias que foram definidas para modelar a variabilidade dos componentes arquiteturais.

Assim, na Seção 4.1 está descrito o primeiro passo da engenharia de domínio, que foi a construção do diagrama de casos de uso. Na Seção 4.2 está descrito o comportamento geral de um sistema tutor inteligente. Na Seção 4.3 está descrito o modelo de *features*, neste modelo encontram-se as variabilidades mapeadas no sistema. Na Seção 4.4 encontra-se descrito o processo de modelagem da arquitetura. Na Seção 4.5 está descrita a arquitetura completa da linha de produto proposta nesta tese. Na Seção 4.6 está descrita a representação semântica da linha de produto. E por fim, na Seção 4.7 está descrito o mecanismo que proporciona a evolução automatizada da linha de produto.

4.1 Diagrama de Casos de Uso

A modelagem dos casos de uso da linha de produto de software proposta é documentada segundo a especificação proposta por [43], assim ela consiste de duas partes: o diagrama de casos de uso, na qual fornece uma representação gráfica dos cenários de uso dos atores envolvidos, além da descrição dos casos de uso, que documenta uma descrição detalhada dos casos de uso envolvidos no sistema. O diagrama de casos de uso da linha de produto de software para sistemas tutores inteligentes contém dois atores: Estudante e Sistema, e ela está completamente descrita na Figura 4.1. Cada caso de uso do sistema é identificado por um nome único. Para ilustrar a documentação dos casos de uso do sistema, as subseções a seguir explicam as descrições de dois casos de uso distintos.

4.1.1 UC25 - Visualizar Relatório de Aprendizagem

A Figura 4.2 ilustra o caso de uso de Visualizar Relatório de Aprendizagem. Tal caso de uso possui quatro casos de uso de extensão (todos opcionais): Visualizar Histórico de Resoluções, Visualizar Desempenho do Domínio, Visualizar Nível de Conhecimento e Visualizar Desempenho no Curriculum.

A Figura 4.3 mostra a descrição do caso de uso Visualizar Relatório de Aprendizagem. A linha 3 da Descrição Narrativa do caso de uso da Figura 4.3 contém um <*visualizar relatório*>, isso quer dizer, que existem diferentes formas de realizar a visualização desse relatório.

4.1.2 UC37 - Tutorar

A Figura 4.4 ilustra o caso de uso Tutorar. Este caso de uso é responsável por realizar o tutoramento do estudante, através do envio de recursos educacionais. Além disso, ele deve verificar se houve um avanço de curriculum por parte do estudante e pode ou não definir uma nova estratégia pedagógica mais adequada para um determinado estudante. Dessa forma, ele possui duas inclusões: Enviar Recurso e Verificar Avanço de Curriculum. O caso de uso que representa a definição da estratégia baseada em teoria de aprendizagem não está ilustrado na Figura 4.4.

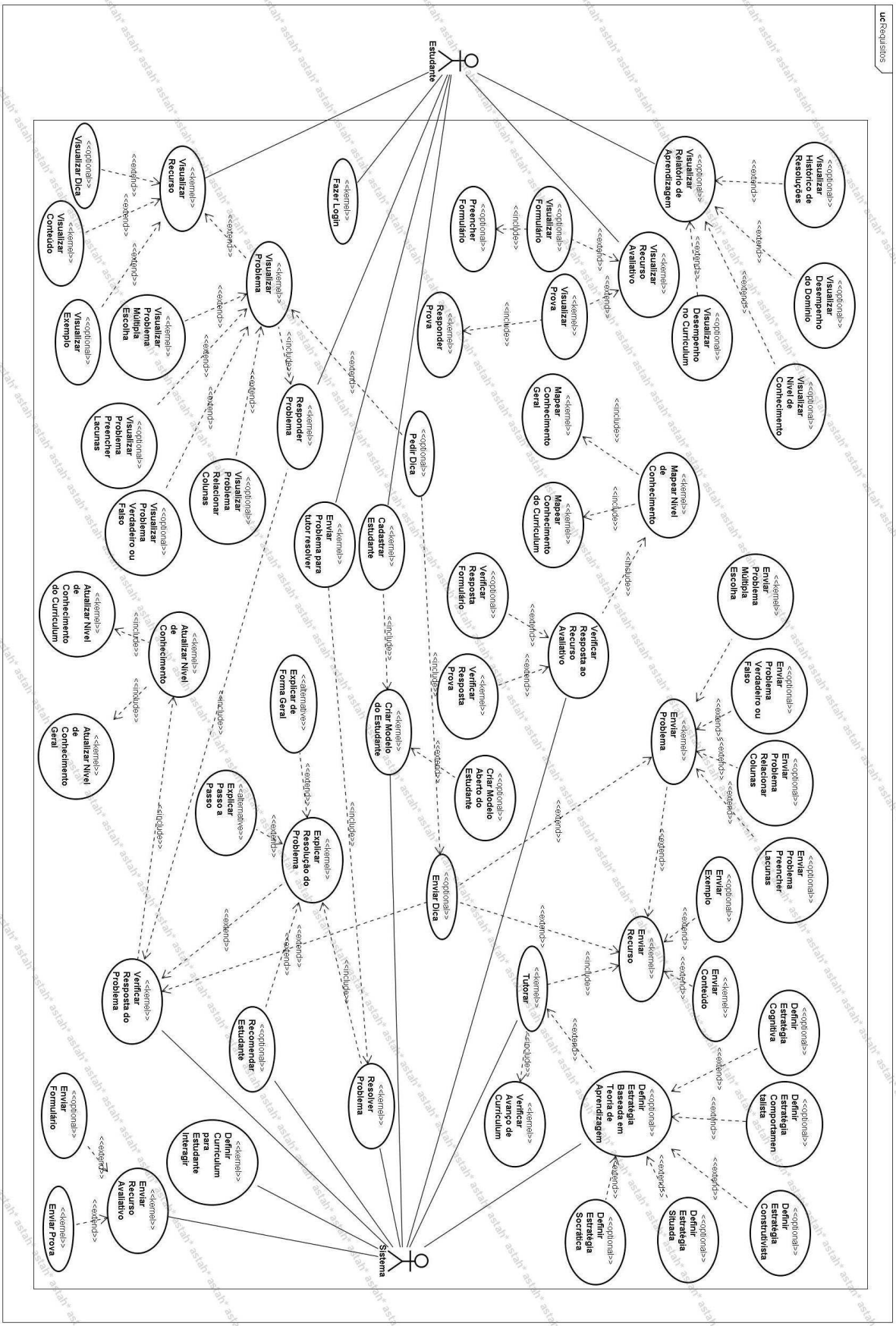


Figura 4.1: Diagrama de Casos de Uso com variabilidade

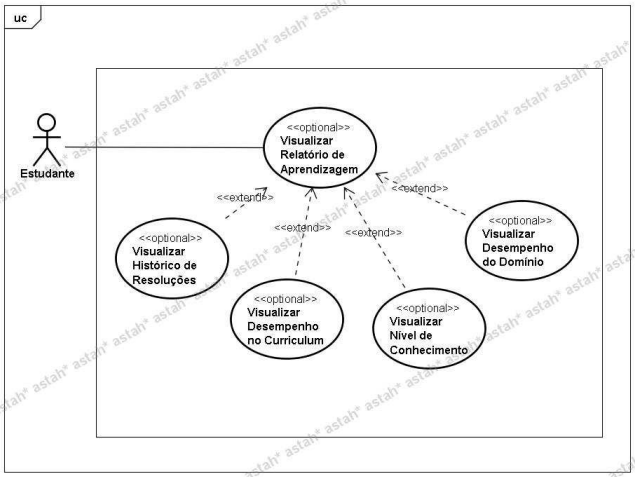


Figura 4.2: Diagrama de casos de uso - UC25 - Visualizar Relatório de Aprendizagem

A descrição do caso de uso Tutorar é mostrada na Figura 4.5. Sua descrição narrativa contém duas linhas que indicam a inclusão de outros dois casos de uso: *Inclui Enviar Recurso* e *Inclui Verificar Avanço de Curriculum.*

1. Nome do Caso de Uso
UC25 – Visualizar Relatórios de Aprendizagem
2. Categoria de Reuso
Optional
3. Resumo
Estudante pode visualizar um relatório de aprendizagem por vez
4. Ator
Estudante
5. Dependências
6. Pré-condições
Estudante não tem mais curriculums para estudar ou ele acabou de terminar de estudar um curriculum.
7. Descrição Narrativa
1. Estudante clica no ícone Visualizar Relatórios de Aprendizagem 2. Estudante escolhe qual relatório ele quer visualizar 3. <visualizar relatório>
8. Pós-condições
Estudante visualizou o relatório de aprendizagem desejado

Figura 4.3: Descrição do caso de uso - UC25 - Visualizar Relatório de Aprendizagem

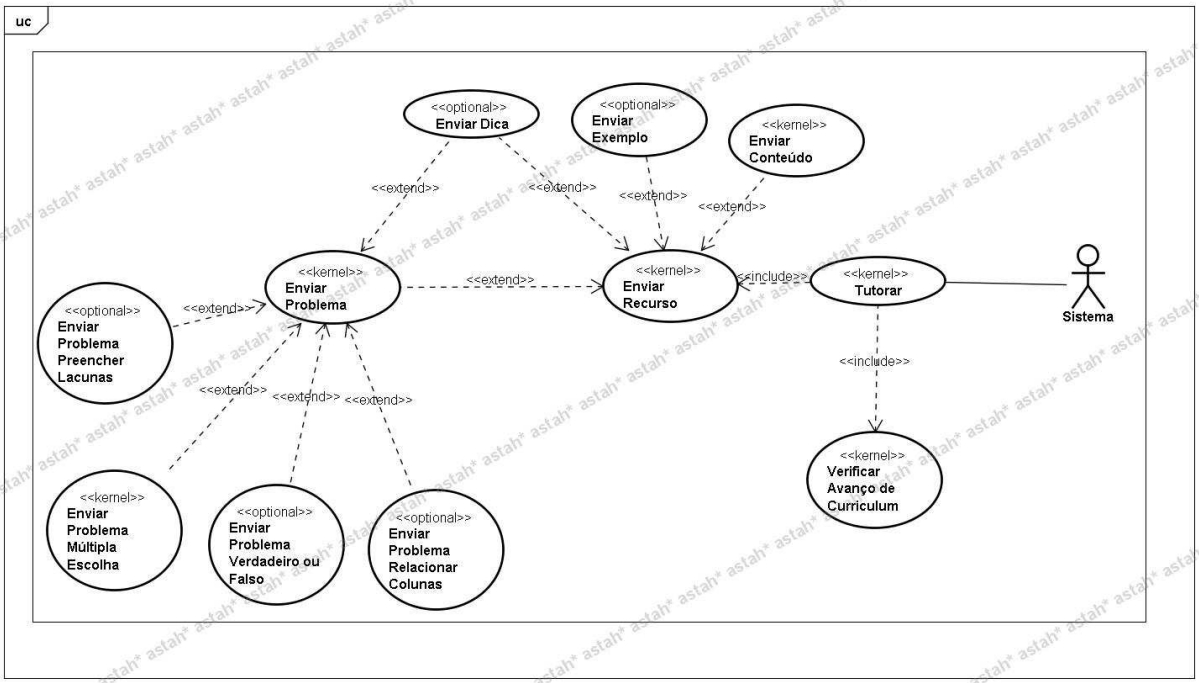


Figura 4.4: Diagrama de casos de uso - UC37 - Tutorar

1. Nome do Caso de Uso
UC37 – Tutorar
2. Categoria de Reuso
Kernel
3. Resumo
Sistema controla o envio de recursos e se o estudante avançou de curriculum
4. Ator
Sistema
5. Dependências
Inclui Enviar Recurso, Inclui Verificar Avanço de Curriculum
6. Pré-condições
Já foi definida uma estratégia pedagógica para o tutoramento, e o sistema sabe o seqüenciamento dos recursos que tem que enviar.
7. Descrição Narrativa
1. Sistema recupera o primeiro recurso que deve enviar através do modelo do domínio 2. *Inclui Enviar Recurso* 3. *Inclui Verificar Avanço de Curriculum*
8. Pós-condições
Estudante recebeu todos os recursos do seqüenciamento.

Figura 4.5: Descrição do caso de uso - UC37 - Tutorar

4.2 Diagrama de Atividades

O diagrama de atividades da Figura 4.6 mostra o funcionamento de forma sequencial de um Sistema Tutor Inteligente independente de domínio[1].O foco principal desse artefato de requisitos é documentar o fluxo de atividades referente à execução de um Sistema Tutor Inteligente. A seguir são explicadas as fases que envolvem o fluxo de atividades contido no diagrama.

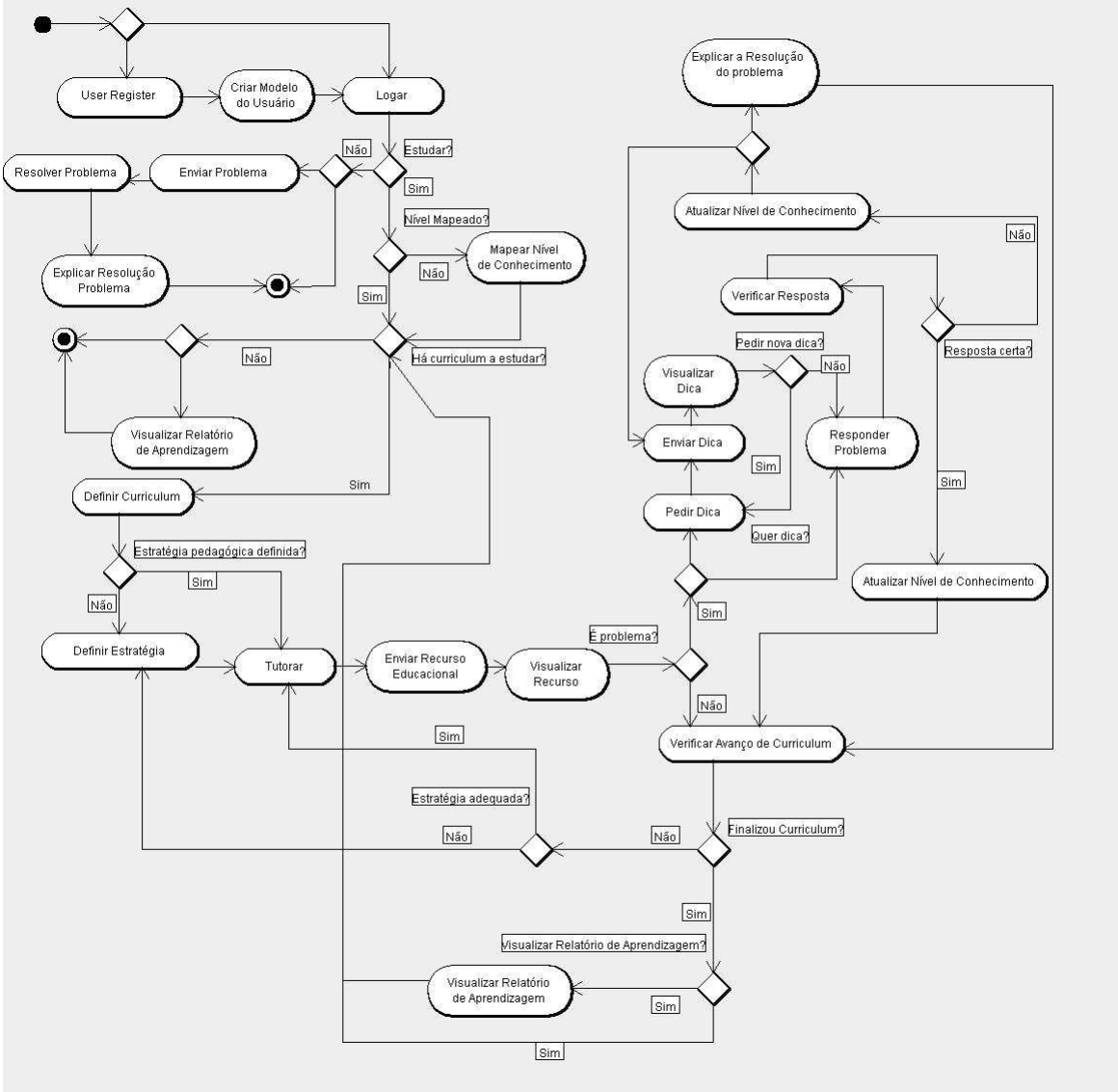


Figura 4.6: Diagrama de Atividades do funcionamento de um STI

[1]Vale ressaltar que o diagrama de atividades aqui proposto abstrai as variantes existentes para pontos de variação do sistema, dessa forma não documenta a variabilidade de forma explícita nesse diagrama.

1. Estudante se cadastra no sistema.

2. Logo após o estudante se cadastrar, o sistema cria o seu respectivo modelo do estudante.

3. O estudante, após o cadastro, pode logar no sistema. A partir desse momento, qualquer interação entre o estudante e o sistema só poderá ser realizado com o estabelecimento do *login*. Isso se faz necessário porque o sistema precisa identificar todo o comportamento do estudante em todos os momentos que o mesmo estiver interagindo com o sistema.

4. O estudante que estiver logado no sistema tem a opção de decidir quando ele começará a estudar.

5. Se ele decidir não estudar, ele pode enviar um problema já cadastrado no sistema para o tutor resolver (mostrando a explicação da resolução do problema), ou ele pode finalizar a interação com o sistema.

6. Se ele quiser estudar, o sistema verificará se o seu nível de conhecimento já foi mapeado, caso contrário, o mesmo iniciará o processo de modelagem do estudante.

7. Após o nível de conhecimento do estudante ser mapeado, o sistema deve verificar se há curriculum para o estudante estudar. Se existir curriculum para ele estudar, o sistema deve definir qual curriculum é indicado para aquele momento do ensino. Caso contrário, ele terá somente a opção de visualizar os relatórios de sua aprendizagem ou finalizar a interação.

8. Depois de definido o curriculum que o estudante vai aprender, será verificado pelo sistema se uma estratégia pedagógica já foi definida para o ensino, e se não foi deve ser definida.

9. Em seguida, inicia-se o tutoramento através do envio de um recurso educacional para o estudante. O recurso a ser enviado vai ser definido pela estratégia pedagógica para o curriculum que o estudante se encontra.

10. O estudante deve visualizar o recurso e se o mesmo for um problema ele tem a opção de pedir quantas dicas ele puder para ajudá-lo a responder o problema. O estudante também pode responder o problema sem pedir dica.

11. Quando o estudante responde o problema, sua resposta deve ser corrigida. Se a resposta estiver correta, o nível de conhecimento do estudante deve ser atualizado. Se a resposta estiver errada, o nível de conhecimento também deve ser atualizado, e dependendo da estratégia pedagógica pode ser enviada uma dica para auxiliar o estudante a responder o problema novamente ou pode ser explicada a resolução do problema.

12. Em seguida deve ser verificado pelo sistema se o estudante avançou de curriculum. Se o curriculum que o estudante estava estudando não foi finalizado, o sistema deve avaliar a definição de uma nova estratégia pedagógica e recomeçar o tutoramento através do envio de novos recursos. Mas se o curriculum foi finalizado, o estudante pode visualizar os relatórios de sua aprendizagem até o momento e voltar para as atividades descritas no passo 2 ou ir direto para as atividades do passo 2.

4.3 Modelo de *Features*

O modelo de notação utilizado para ilustrar o modelo de features foi o proposto em [51].A Figura 4.7 ilustra o modelo de *features* da linha de produto de sistema tutores inteligente na notação indicada. Para este trabalho, o primeiro passo dentro da modelagem da linha de produto proposta, foram estabelecidas as seguintes relacionamentos entre as *features* presentes no diagrama representado na Figura 4.7. Suas respectivas justificativas para cada associação seguem a seguir:

Todos os STIs, desta proposta, contém obrigatoriamente o método de ensino baseado na resolução de problema. Contudo, existem variações de como esse método atuará em cada tutor. Essas variação ocorrerão através das escolhas das seguintes *features*:

- *General:* É uma *feature* que indica que a explicação de um determinado problema será realizada de maneira geral, ou seja, uma única explicação sobre a teoria envolvida para a resolução de cada problema;

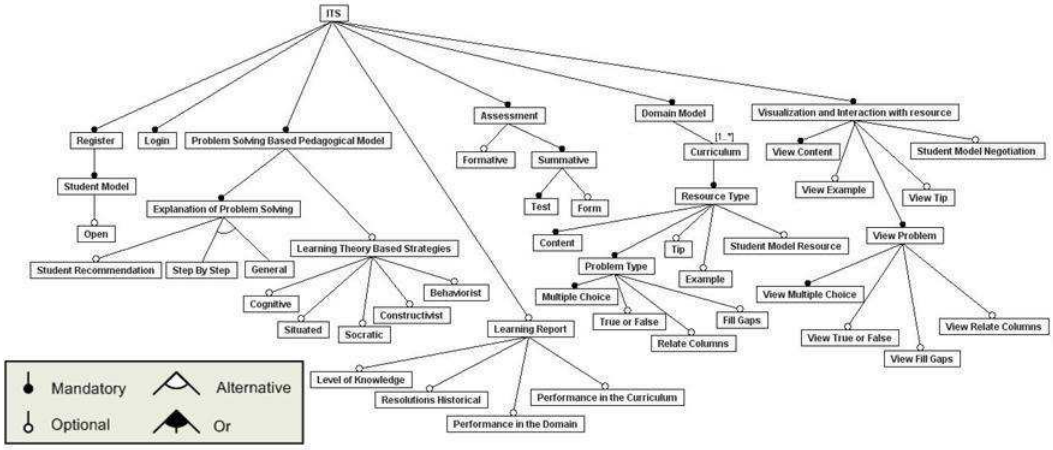


Figura 4.7: Diagrama de *Features*

- *Step by Step:* Indica que a explicação será realizada através da demonstração de como o problema é resolvido passo a passo. Então, dessa forma, um aprendiz tem a oportunidade de verificar todo o processo da resolução de cada problema.

- *Student Recomendation:* É um tipo de explicação onde o estudante tem a possibilidade de entrar em contato com outros estudantes para o ajudar no entendimento da resolução de problema, oferecendo uma perspectiva diferente de explicação.

Também deve ser obrigatório em todo sistema tutor inteligente um modelo do domínio que descreve o domínio que vai ser ensinado para o estudante. Dessa forma, todo modelo do domínio deve conter vários curricula. Além disso, todo curriculum deve possuir recursos. Existem dois tipos de recursos que devem ser obrigatórios para todos os sistemas tutores inteligentes: Conteúdo e Problema. Existem também três tipos de recursos opcionais para as aplicações baseadas na linha de produto: Dica, Exemplo e Modelo do Estudante. Para o tipo de recurso problema, especificamente, o diagrama mostra que o problema do tipo múltipla escolha é obrigatório para todas as aplicações, por outro lado os problemas de verdadeiro ou falso, preencher lacunas e relacionar colunas são opcionais.

4.4 Modelagem da Arquitetura

Como foi mostrado na Seção 2.5, o processo *UML Components* adota uma arquitetura pré-definida em quatro camadas. Dessa forma, a arquitetura da linha de produto para STIs já

começa com as quatro camadas definidas por esse processo. O foco da arquitetura está nas duas camadas principais de Sistema e Negócio.

Os dois principais artefatos de entrada para a especificação dos componentes são: diagrama de casos de uso do sistema e modelo conceitual do negócio. Na Seção 2.5, foi explicado também que os casos de uso são usados para identificar as interfaces da camada do sistema e o modelo conceitual do negócio é usado para identificar as interfaces da camada de negócio. Além disso, a especificação dos componentes que compõem a arquitetura, seguindo o processo *UML Components*, é dividida em três etapas: (i) Identificação dos componentes; (ii) Interação entre os componentes; e (iii) Especificação final. A especificação final é um estágio opcional para a definição da arquitetura (apesar de ser recomendado por [20]), envolvendo a especificação formal dos componentes. Para o escopo deste trabalho, somente as duas primeiras etapas do processo foram seguidas, e estão presentes nas duas subseções seguintes.

4.4.1 Identificação dos Componentes

Os principais artefatos produzidos nessa fase são três: (i) especificação das interfaces do sistema; (ii) identificação das interfaces de negócio; e, (iii) identificação dos componentes.

Especificação das Interfaces do Sistema

Pelo processo *UML Components*, cada caso de uso do diagrama vira uma interface da camada do sistema, e as operações de cada interface são identificadas analisando a descrição narrativa do caso de uso em questão. Devido à quantidade de casos de uso existentes no sistema, e consequentemente a quantidade de interfaces do sistema identificadas através de tais casos de uso, nesta fase da especificação foi escolhido o caso de uso **UC37-Tutorar**, descrito na Seção 4.1.2, para demonstrar como foi feita a especificação das interfaces do sistema.

Dessa forma, inicialmente cada caso de uso envolvido com *Tutorar* vira uma interface do sistema. Como mostrado na Figura 4.8.

Definidas as interfaces do sistema, o próximo passo é definir as operações de cada interface baseado na descrição detalhada de cada caso de uso. Nesta fase de especificação, as operações não contêm parâmetros nem tipo de retorno.

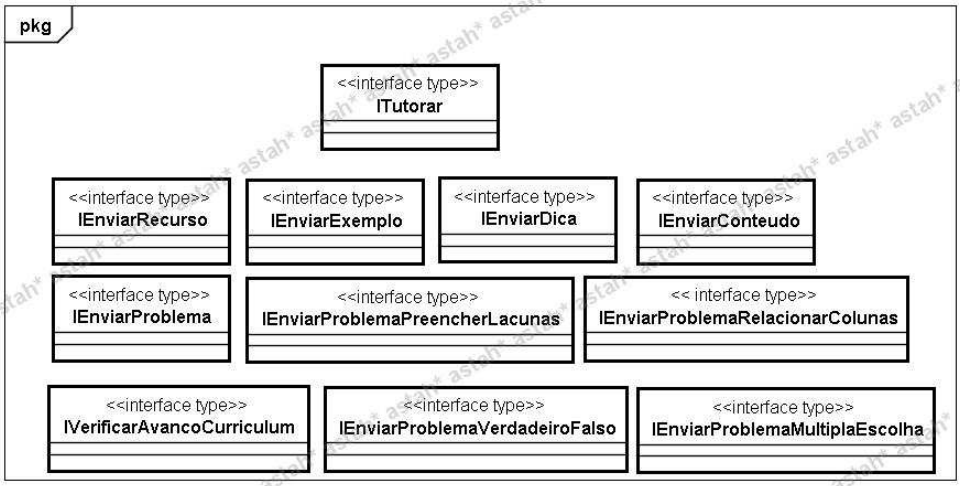


Figura 4.8: Interfaces do sistema dos casos de uso envolvidos com o caso de uso UC37 - Tutorar

Como o caso de uso **UC37-Tutorar** inclui os casos de uso **UC38-Enviar Recurso** e **UC47-Verificar Avanço de Curriculum**, e como os casos de uso incluídos não são muito extensos, foi definido que eles virariam operações da interface *ITutorar*, ao invés de possuírem uma interface de sistema própria.

Além disso, a definição das operações da interface *ITutorar* também deve permitir a flexibilidade da escolha do tipo de recurso (variabilidade do sistema) que vai ser enviado. Sabe-se que existem vários tipos de recursos que podem ser enviados, uns obrigatórios (*kernel*) e outros opcionais (*optional*). Logo, a primeira decisão tomada para modelar variabilidade na arquitetura da linha de produto baseada em *UML Components* foi investigar se existe um caso de uso que é extendido por diferentes casos de uso que executam tal caso de uso de uma forma diferente. Nos casos positivos, deve ser criada uma operação "abstrata"na interface do caso de uso que possui um parâmetro de configuração que delega a operação para o caso de uso extendido adequado, e cada caso de uso que o extende deve implementar esta mesma operação para o seu tipo específico.

A Figura 4.9 mostra as novas interfaces do sistema, com as respectivas operações. A operação ***gerarPaginaRecurso(tipo : Recurso)*** é que contém o parâmetro de configuração que vai controlar o tipo de recurso que vai ser utilizado. Nota-se também, na Figura 4.9, que as interfaces *IEnviarRecurso* e *IEnviarProblema* definidas anteriormente foram extintas, isso aconteceu pois o *ITutorar* é responsável por controlar o tipo de recurso que vai ser escolhido.

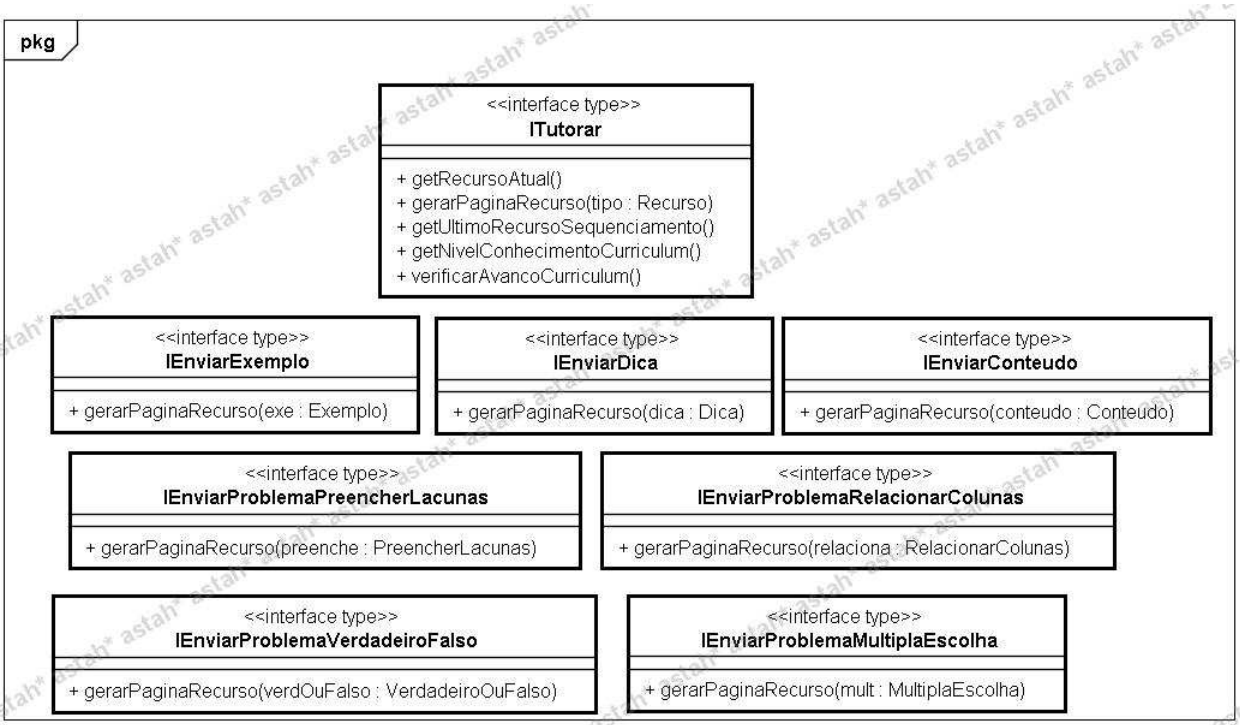


Figura 4.9: Interfaces do sistema dos casos de uso envolvidos com o caso de uso UC37-Tutorar com as operações definidas

Identificação das Interfaces de Negócio

As interfaces de negócio são abstrações das informações que devem ser gerenciadas pelo sistema [20]. O processo para identificá-las é como segue:

1. Criar o Modelo de Tipo de Negócio - Este modelo é basicamente igual ao modelo conceitual do negócio mostrado na Seção 2.4, só que o de tipo de negócio contém informações específicas do negócio que devem ser guardadas pelo sistema que está sendo especificado. Ele inclui mais detalhes dos atributos de cada tipo que constitui o modelo;

2. Identificar *Core Business Types* - O propósito de identificar os tipos de negócio "centrais" é para começar a pensar sobre qual informação é dependente de outra informação e quais informações podem ser independentes;

3. Criar as interfaces para os tipos centrais (*core types*) e adicioná-las ao modelo de tipo de negócio;

4. Refinar o modelo de tipo de negócios para indicar as responsabilidades das interfaces de negócio.

Cada passo explicado acima foi seguido para especificar as interfaces de negócio para a linha de produto proposta. A Figura 4.10 ilustra o modelo de tipo de negócio que foi criado para o domínio de STIs. As classes desse modelo contêm alguns atributos que os detalham onde cada tipo do modelo é identificado por um estereótipo ou do tipo *«type»* ou do tipo *«core»*.

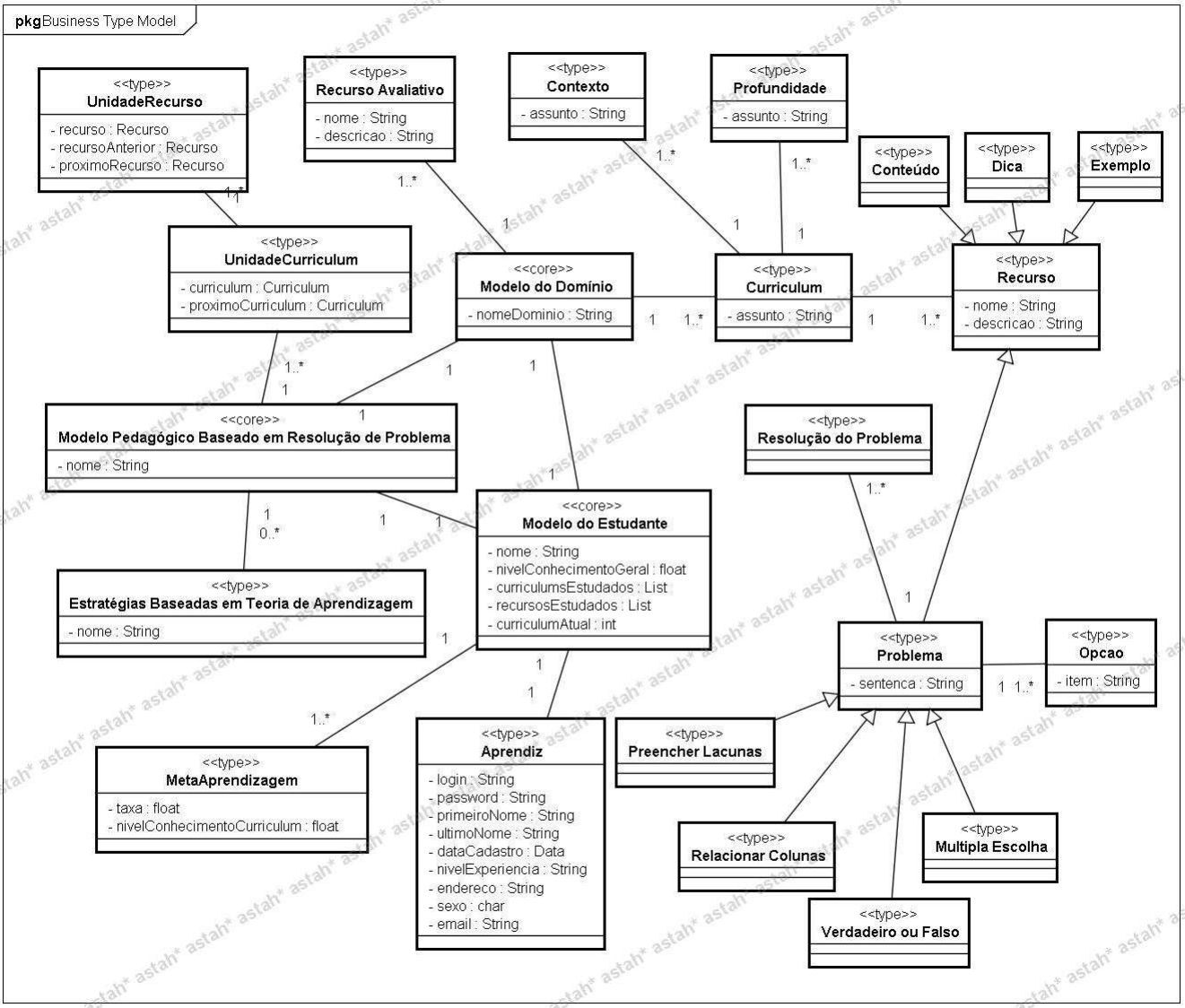


Figura 4.10: Modelo de Tipo de Negócio

Através do modelo de tipo de negócio definido, foram definidos quais desses tipos são centrais (*core types*). Por exemplo, para ser um tipo "central", o tipo de negócio deve ter

existência independente no negócio, e isso é caracterizado por duas características: um identificador do negócio, geralmente independente de outros identificadores e existência independente, ou seja, sem associações obrigatórias (isso quer dizer que ele não está contido em nenhum outro tipo).

No contexto de STI, foi definido que os tipos "centrais"são os quais representam os três modelos: (i) Modelo Pedagógico; (ii) Modelo do Domínio e (iii) Modelo do Estudante. Dessa forma, os tipos Modelo Pedagógico Baseado em Resolução de Problema, Modelo do Domínio e Modelo do Estudante têm seu estereótipo alterado para *«core»*. Todos os outros tipos de negócio provêm detalhes dos tipos "centrais"e possuem o estereótipo *«type»*.

Seguindo o processo descrito acima, deve ser criada uma interface para cada *core type*, adicioná-las ao modelo de tipo de negócio e indicar as responsabilidades das interfaces. A Figura 4.11 ilustra as interfaces associadas com os *core types* adicionadas ao modelo de tipo de negócio previamente definido. A seguir são descritas as responsabilidades das interfaces associadas com os *core types*:

- *IModeloDominioMgt* - O modelo do domínio contém todos os curricula do domínio que vai ser ensinado, além disso ele deve ter acesso aos recursos avaliativos do domínio. Dessa forma, a interface *IModeloDominioMgt* tem responsabilidade sobre os tipos de negócio Curriculum e Recurso Avaliativo;

- *IModeloEstudanteMgt* - O modelo do estudante deve manter informações do estudante e também deve manter informações das metas de aprendizagem daquele estudante para cada curriculum do domínio. Logo, a interface *IModeloEstudanteMgt* tem responsabilidade sobre os tipos de negócio Aprendiz e MetaAprendizagem;

- *IModeloPedagogicoMgt* - O modelo pedagógico deve manter informações do sequenciamento dos recursos para um determinado curriculum, além disso ele também deve conhecer a estratégia pedagógica que define tal sequenciamento. Portanto, a interface *IModeloPedagogicoMgt* tem responsabilidade sobre os tipos de negócio UnidadeCurriculum e Estratégias Baseadas em Teoria de Aprendizagem.

Dadas as identificações das interfaces do negócio e das suas respectivas responsabilidades, a fase de especificação das interfaces de negócio está concluída.

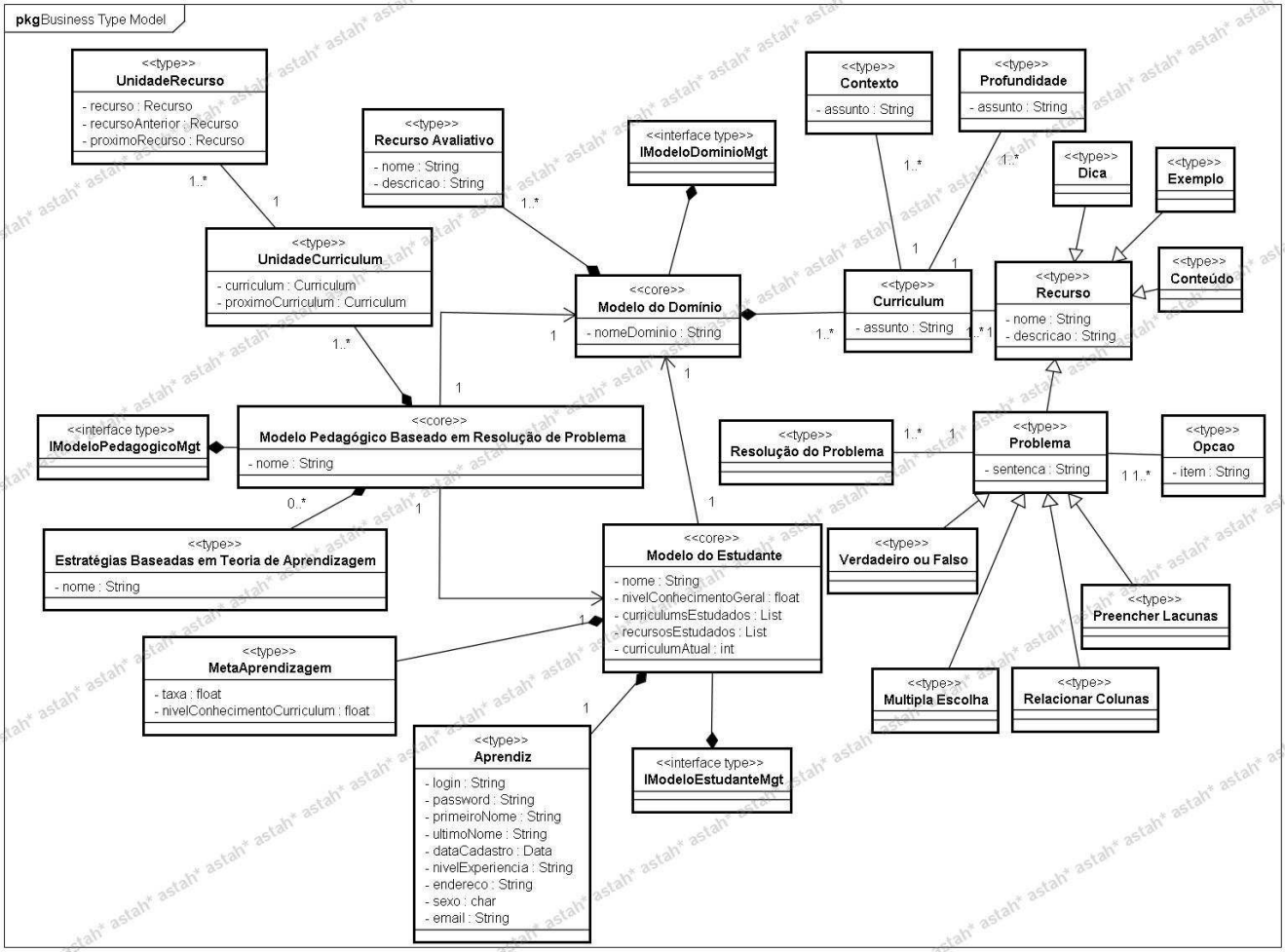


Figura 4.11: Diagrama de responsabilidade das interfaces do modelo de tipo de negócio

Identificação dos Componentes

Depois de identificadas as interfaces de sistema e negócio é chegada a hora de identificar os componentes que implementam tais interfaces. Segundo o processo *UML Components*, a ideia inicial, neste estágio do processo, é que cada interface tanto de sistema quanto de negócio tenha um novo componente associado.

Dessa forma, para ilustrar a criação dos componentes da camada de sistema, voltamos para os casos de usos envolvidos com o caso de uso **UC37-Tutorar**. Identificamos as seguintes interfaces para este subsistema: *ITutorar*, *IEnviarExemplo*, *IEnviarDica*, *IEnviar-Conteudo*, *IEnviarProblemaPreencherLacunas*, *IEnviarProblemaRelacionarColunas*, *IEnviarProblemaVerdadeiroFalso* e *IEnviarProblemaMultiplaEscolha*.

Por conseguinte, cada interface deve ter um componente associado. Para o contexto

de linha de produto, essa identificação tem algumas particularidades, por exemplo, para as interfaces supracitadas, sabe-se que *ITutorar* controla a variabilidade do recurso que vai ser enviado, portanto o componente que implementa *ITutorar*, deve requerer todos os diferentes tipos de interfaces dos componentes que implementam as diferentes variantes de recurso.

Dessa forma, do mesmo modo que [43] propôs para os casos de uso, são definidos neste trabalho que os mesmo estereótipos relacionados com os casos de uso devem ser refletidos nos componentes que implementam tais funcionalidades. Logo uma primeira configuração arquitetural pode ser gerada para o subsistema de tutoramento, como mostrada na Figura 4.12.

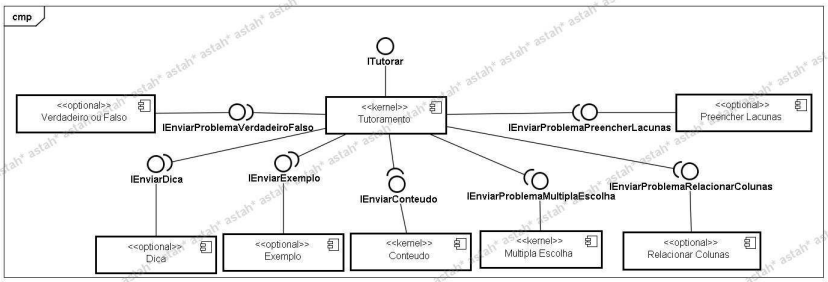


Figura 4.12: Exemplo de Configuração arquitetural na camada de sistema

A identificação dos componentes ilustrada acima deve ser feita para todas as interfaces do sistema. Após isso, também devem ser identificados os componentes que implementam as interfaces de negócio, isso ocorre do mesmo modo que nas interfaces de sistema, um componente para cada interface, como mostrado na Figura 4.13.

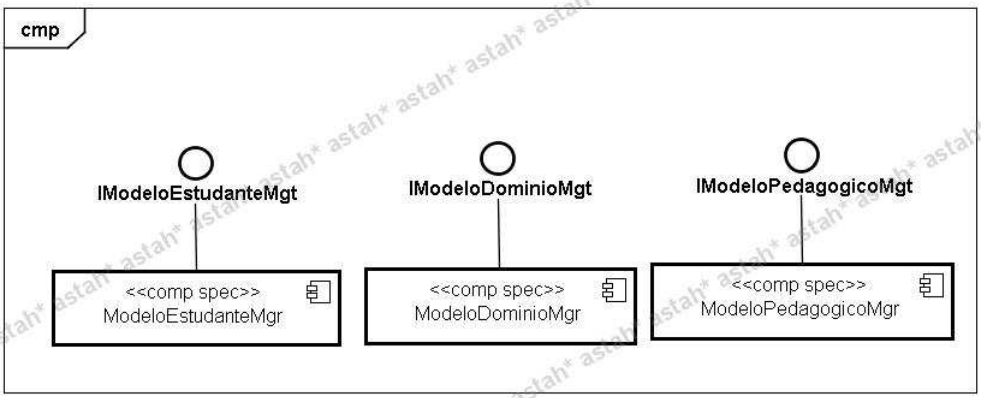


Figura 4.13: Componentes das interfaces de negócio

Como a arquitetura definida é em camadas, a camada de sistema usa as operações da

camada de negócio, dessa forma devem existir associações entre as interfaces das duas camadas arquiteturais. Sabe-se que o tutoramento em um STI deve conhecer o estudante que vai receber os recursos enviados, ou seja, requer informações do modelo do estudante, além disso, deve-se conhecer a interface que tem responsabilidade sobre a estratégia pedagógica do sequenciamento que está acontecendo para poder mudar de estratégia dependendo do processo de ensino, logo requer informações do modelo pedagógico. Enfim, ele também necessita das informações do modelo do domínio, pois ele tem responsabilidade sobre os recursos do domínio.

Portanto a seguinte arquitetura da Figura 4.14 pode ser formada, para as camadas arquiteturais de sistema e negócio.

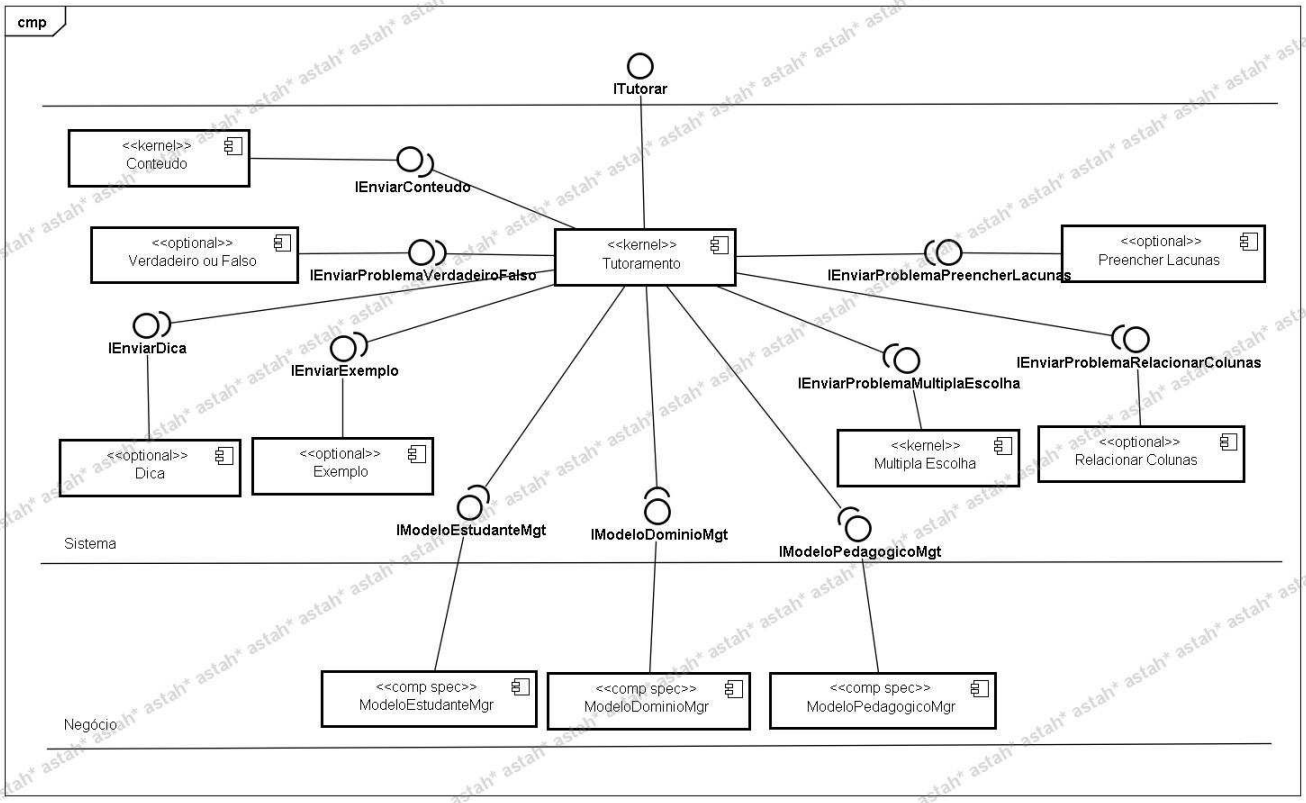


Figura 4.14: Arquitetura do subsistema tutoramento

A mesma análise feita para o tutoramento deve ser feita para todos os outros componentes do sistema, no sentido de identificar quais interfaces providas pelos componentes da camada de negócio, os componentes da camada de sistema devem requerer.

4.4.2 Interação entre os Componentes

Esta fase tem o objetivo principal de definir como os componentes vão funcionar juntos para entregar as funcionalidades requeridas. Esta fase tem duas atividades principais: a descoberta das operações das interfaces de negócio e o refinamento das operações das interfaces de sistema.

Descoberta das Operações das Interfaces de Negócio

Para descobrir as operações das interfaces de negócio, deve-se analisar um ou mais diagramas de colaboração que traçam o fluxo de execução resultante da invocação de cada operação das interfaces do sistema.

O diagrama de componentes da arquitetura mostrado acima demonstra que o tutoramento deve usar as interfaces *IModeloPedagigoMgt*, *IModeloDominioMgt* e *IModeloEstudanteMgt*. Dessa forma, as operações das interfaces de negócio basicamente são as mesmas das interfaces de sistema que utilizam suas informações. O diagrama de sequência da Figura 4.15 ilustra mais claramente quais operações utilizam informações de quais interfaces de negócio.

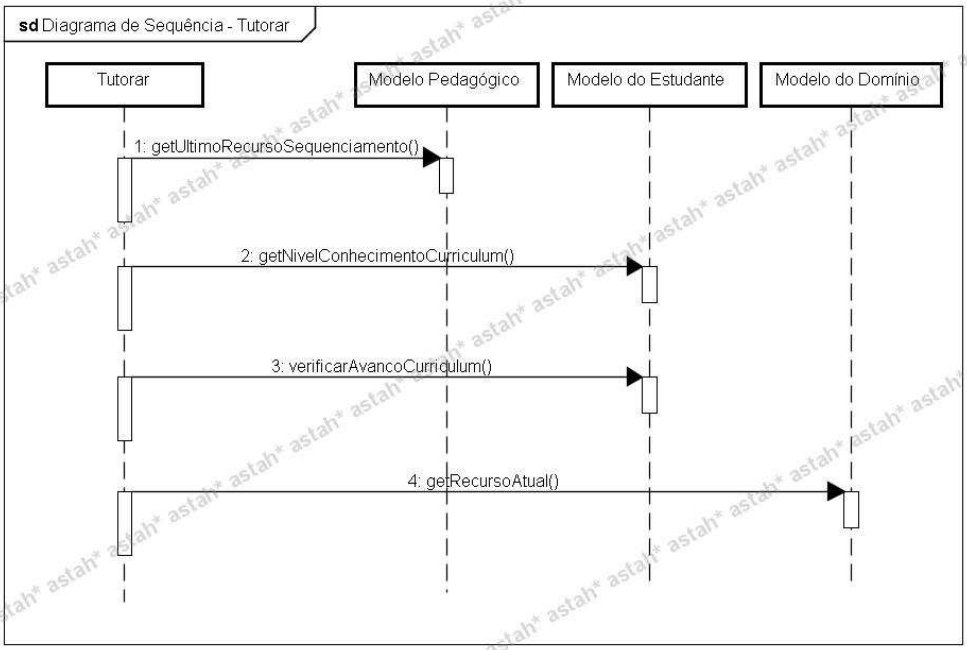


Figura 4.15: Diagrama de Sequência de Tutoramento

A operação da interface de sistema *ITutorar* que utiliza informações do *IModeloPedagicoMgt* é a ***getUltimoRecursoSequenciamento()***. Logo, tal operação é delegada para a interface de negócio *IModeloPedagogicoMgt*. Do mesmo modo, as operações ***getNivelConhecimentoCurriculum()*** e ***verificarAvancoCurriculum()*** que utilizam informações da interface de negócio IModeloEstudanteMgt são repassadas para ela. A operação ***getRecursoAtual()*** que utiliza informações da interface *IModeloDominioMgt* é delegada para ela. A Figura 4.16 ilustra as interfaces de negócio com as respectivas operações identificadas.

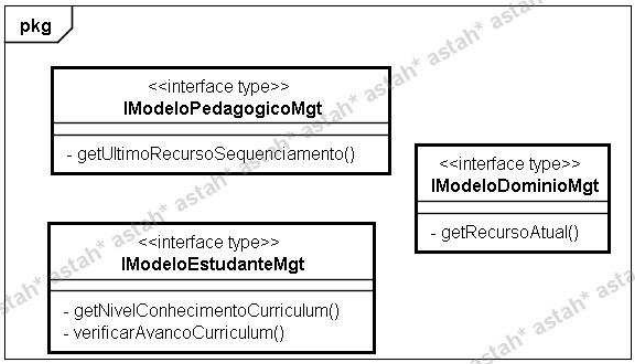


Figura 4.16: Operações das interfaces de negócio

A descoberta das interfaces de negócio é realizada através do mesmo procedimento feito acima, ao final todas as operações das interfaces de negócio estão definidas.

Refinamento das Operações das Interfaces do Sistema

Até esse ponto não se sabe nenhuma assinatura das operações identificadas tanto para as interfaces de sistema quanto de negócio. Esse estágio é responsável por refinar as operações das interfaces de sistema e consequentemente refinar as operações das interfaces de sistema.

Por exemplo, a operação ***getRecursoAtual()*** serve para como foi feito o refinamento, a ideia dessa operação é retornar um recurso do modelo do domínio. Dessa forma deve retornar um objeto que contenha detalhes de tal recurso. Logo, foi criado um tipo de dado (*data type*) chamado *DetalhesRecurso* que modela os dados contidos em um recurso. Portanto, a operação ***getRecursoAtual()*** foi refinada para mostrar o tipo de retorno e o parâmetro que deve ser passado. A Figura 4.17, a seguir, mostra a operação refinada, e a especificação do tipo de dado retornado por ela.

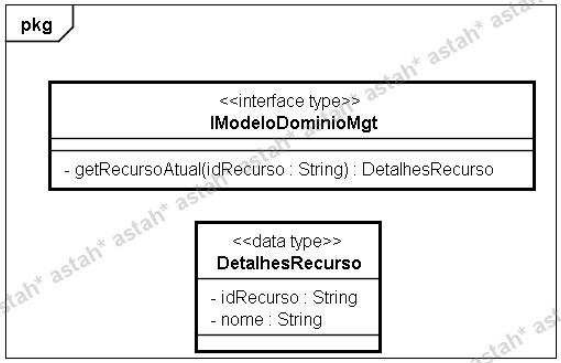


Figura 4.17: Operação ***getRecursoAtual()*** refinada

Todas as operações das interfaces especificadas foram refinadas e os tipos de dados (*data types*) necessários foram criados. A Figura 4.18 ilustra as interfaces do subsistema de tutoramento refinadas.

Ao final desta etapa a Arquitetura da Linha de Produto para STIs já está definida.

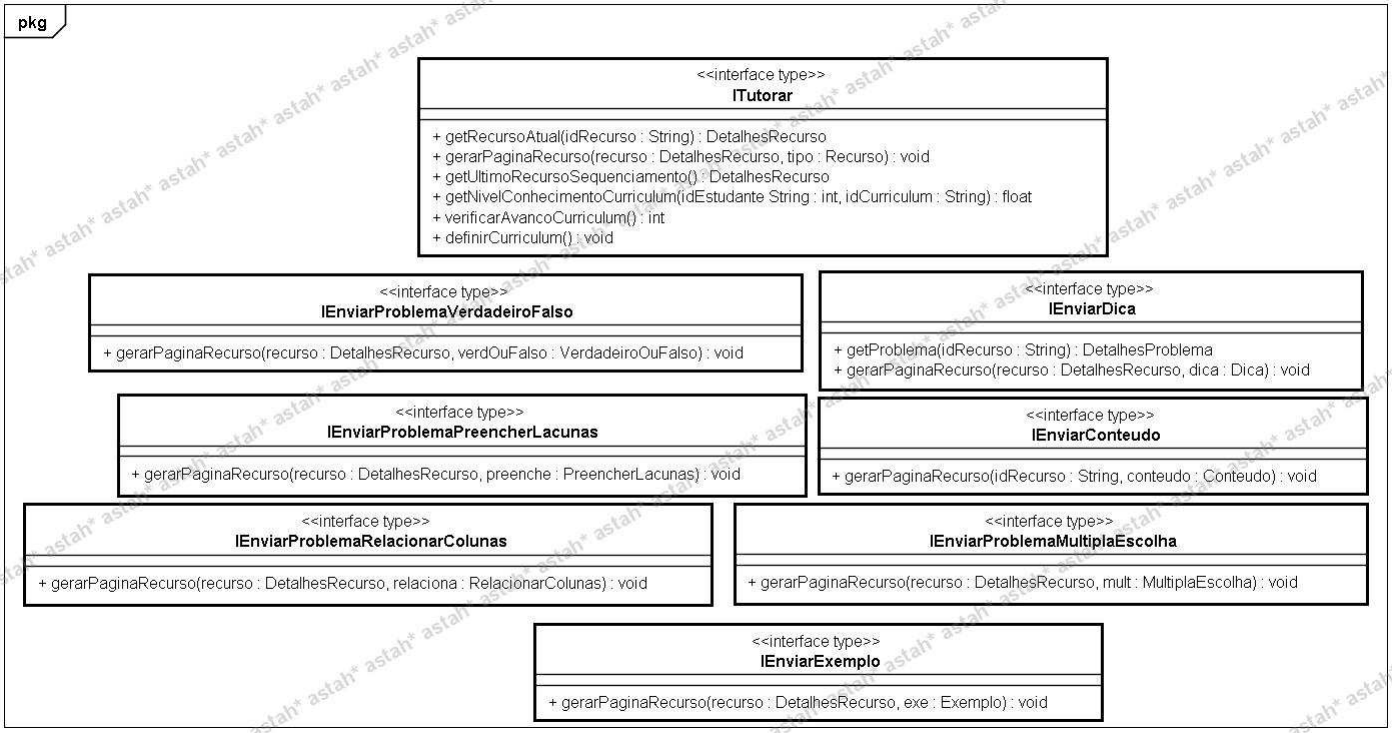


Figura 4.18: Interfaces refinadas do subsistema tutoramento

4.5 Arquitetura da Linha de Produto de Software para STIs

Os artefatos dos componentes das camadas pertencentes ao lado servidor, que serão apresentados nas próximas subseções. de forma detalhada.

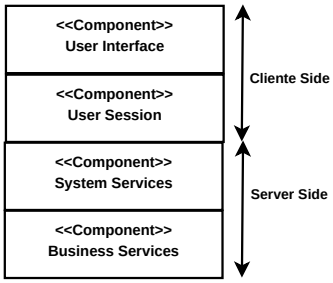


Figura 4.19: UML Components Architecture

4.5.1 Principais Artefatos

Os principais artefatos estão descritos na Figura 4.20, onde o principal componente é o *Tutoring*. Nesta figura estão descritos os três únicos componentes da camada de negócios, conforme a seguir:

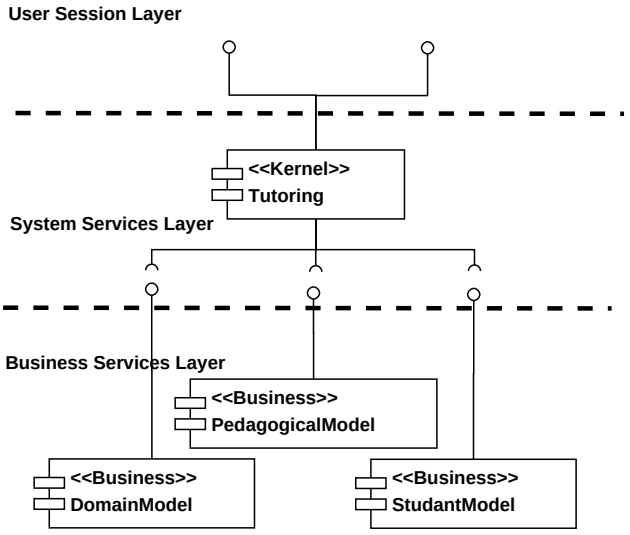


Figura 4.20: Artefatos Principais da Arquitetura

- *Student Model:* Neste componente estão representados os mecanismos de representação e manipulação do modelo do estudante. Isso se faz necessário em todos STIs, pois, é através deste componente e suas informações que se torna possível prover instrução personalizada nas camadas superiores. Além disso, para manipulação com este componente, sobretudo os componentes da camada System Services, foi especificado uma interface chamada *I-StudentModel*.

- *Domain Model:* Neste componente estão representados os modelos do domínio, juntamente com os seus currículos. Este componente é importante para fazer com que os componentes da camada superiore possam saber o que deve ser ensinado em cada domínio de conhecimento. Além disso, para manipulação com este componente, sobretudo os componentes da camada System Services, foi especificada uma interface chamada *I-DomainModel*.

- *Pedagogical Model:* Neste componente estão representadas as possíveis estratégias pedagógicas e suas respectivas táticas, para que as mesmas possam ser utilizadas em

todo o processo de ensino. Também como os demais componentes desta camada foi especificado uma interface chamada *I-PedagogicalModel*.

Por outro lado, todo o processo de ensino é realizado pelo componente *Tutoring*, que encontra-se na camada *System Services*. Esse componente é responsável por disponibilizar e acompanhar todos os recursos educacionais envolvidos em um processo de ensino de algum domínio de conhecimento. Além disso, esse componente deve tomar suas decisões sempre baseadas em estratégias de ensino específicas, no modelo do estudante e no modelo do domínio. Os componentes da camada *User Session* se comunicam fundamentalmente com ele para realizar todas as suas atribuições, e essa comunicação se dá através das interfaces *I-Tutoring* e *I-EducacionalResource*.

4.5.2 Recursos Educacionais

Conforme foi dito na sub seção 4.5.1, o componente *tutoring* é responsável por disponibilizar o recursos educacionais aos alunos em seu processo de ensino. Os recursos educacionais previstos na linha de produto proposta nesta tese foram todos mapeados em componentes, que estão representados nas Figuras 4.21 e 4.22. A seguir segue a descrição dos componentes que representam os recursos educacionais, juntamente com a descrição de seus sub artefatos:

Subject: Esse é um componente obrigatório que permite a disponibilização de conteúdo para ser fornecido ao estudante. Ele é importante para que o estudante possa fazer um estudo teórico ao conteúdo que o mesmo vai se submeter a aprender. Então, este componente tem como responsabilidade armazenar os conteúdos e disponibilizá-los conforme as necessidades de cada estudante. A solicitação de um conteúdo é feita de maneira indireta, onde o componente *Tutoring* repassa as condições e necessidades de um determinado estudante, e ele recomendará um conteúdo adequado para essas condições recebidas. Uma outra responsabilidade é justamente enviar a exibição desse conteúdo, que dependerá muitas vezes do dispositivo que o aluno em questão estiver estudando no momento. Essas interações entre o componente *Tutoring* e o componente *Subject* são realizadas através das interfaces providas pelo Subject, que são *I-ShowSubject* e *I-SendSubject*;

Example: É um conteúdo opcional que é responsável por fornecer conteúdos relacionados a exemplos ou analogias para complementar conteúdos. Na realidade, esse tipo de conteúdo pode não ser importante para alguns domínios, como história, por exemplo. Por outro lado, ele pode ser um mecanismo muito importante como nos domínios da matemática e física, pois estes precisam de exemplos para fazer com que o aluno possa fixar o conhecimento. Como componente, o procedimento para que o componente *Tutoring* possa interagir com ele é análogo ao *Subject*, que é feito pelas interfaces providas *I-ShowSubject* e *I-SendSubject*. A interface *I-SendSubject* representa a forma como o componente *tutoring* vai solicitar um determinado exemplo, que é feito com parâmetros semelhantes ao do conteúdo, e a interface *I-ShowExample* tem a mesma função de mostrar uma maneira de exibir o exemplo;

Hint: Uma dica é um tipo de recurso educacional que precisa ser processado em tempo real, pois, ela é requisitada justamente em tempo real, ou seja, quando o estudante está interagindo com o sistema e o mesmo está com algum tipo de problema. Por isso que a mesma precisa obter informações do modelo do estudante e do modelo do domínio sempre que a mesma receber alguma solicitação. Pois, para cada requisição recebida, faz-se necessário que o mesmo leve em consideração o estado atual do estudante em relação ao problema que o mesmo estiver sendo submetido, e ao modelo do domínio para retornar com uma resposta adequada. Como componente, o mesmo pode receber solicitação sob duas perspectivas, a primeira é saber a dica adequada para o estudando em tempo real de sua interação, através da interface *I-SendHint*, e mostrar como ele deve ser exibido, através da interface *I-ShowHint*.

Por outro lado, a forma que o sistema tem de realmente avaliar se o estudante está progredindo ou não em relação ao conhecimento, é no envio dos problemas e seu respectivo acompanhamento. Na linha de produtos proposta nessa tese, está prevista a implementação de quatro possíveis tipos de problemas (conforme Figura 4.22): Verdadeiro ou Falso; Preencher Lacunas; Relacionar Colunas; Múltipla Escolha. Cada um desses componentes que representam um tipo de pergunta possui duas interfaces providas *I-Send* e *I-Show*. A Interface *Send* é, em todos os casos, para enviar um problema para a demanda submetida em tempo real pelo componente *Tutoring*. Já a interface *I-Show* é útil para enviar como um dos

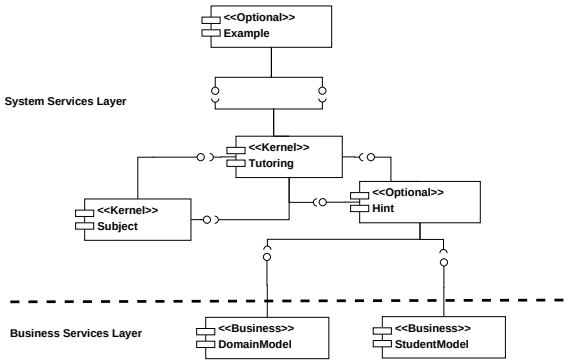


Figura 4.21: Artefatos que processam os recursos educacionais.

seus problemas deve ser exibido.

Na realidade, o componente *problem*, isoladamente descrito na Figura 4.23, é quem vai ser responsável pelo envio do problema ao aprendiz. Esse componente é responsável também por fazer a avaliação de utilização dos problemas, de forma que o mesmo possa ajustar a relação do problema e sua eficácia em relação ao modelo do domínio e estudante.

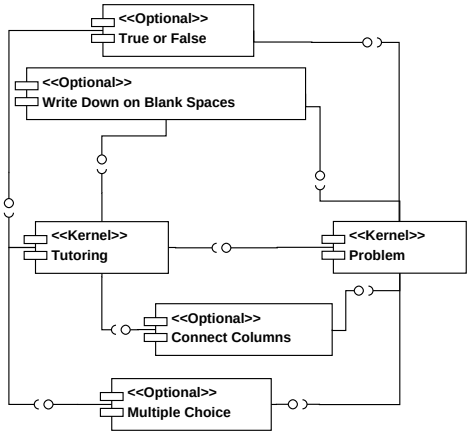


Figura 4.22: Artefatos de Tipos de Problema

4.5.3 Relatórios

Na Figura 4.24 estão descritos os componentes de relatórios analíticos inerentes aos resultados dos comportamentos dos aprendizes com os recursos educacionais. Estes relatórios são importantes para que os autores dos STIs possam analisar o quanto os estudantes estão evoluindo, identificar quais são os currículos que geram mais dificuldades no aprendizado do aluno, identificar quais são os recursos educacionais mais eficientes e verificar quais são

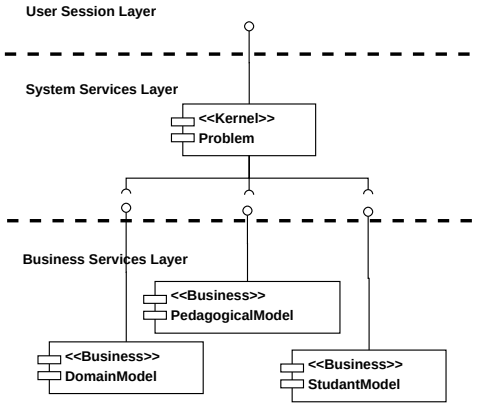


Figura 4.23: Artefatos que Processa os Problemas

as estratégias pedagógicas que estão melhor se comportando.

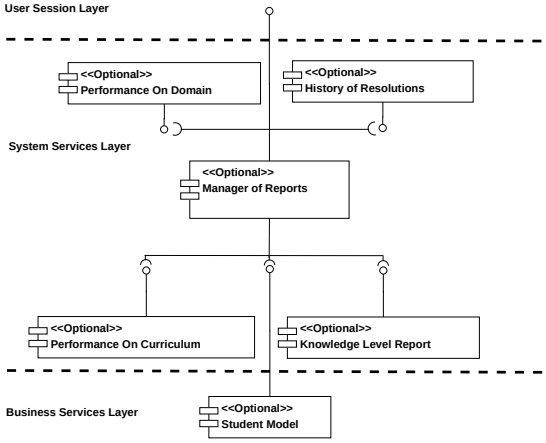


Figura 4.24: Artefatos de Relatórios

O Componente principal desse trecho (Figura 4.24) da arquitetura é o *Manager Of Report*, pois é ele quem prover uma interface para os componentes da camada *User Session*, onde sua interface se chama *I-ManagerOfReport*. Além disso, ele tem a responsabilidade de conhecer e manipular as interfaces dos componentes que efetivamente realizam o processamento dos dados. Estes componentes são opcionais, e estão descritos a seguir:

Performance On Domain: Esse componente tem a responsabilidade de gerar visualização de dados sobre a performance detalhada dos aprendizes nos domínios. Com esses dados, pode ser verificado como os estudantes se comportam de forma global. Por exemplo, quais estudantes não conseguiram obter um resultado satisfatório no domínio em

um determinado período de tempo ou quais estudantes com determinada características tiverem um resultado muito expressivo.

History Of Resolutions: Esse componente tem a responsabilidade de coletar informações sobre todas as resoluções dos problemas de todos os estudantes e gerar visualizações inerentes a essas informações. Por exemplo, é possível identificar quais são os aspectos de cada tipo de problema que possuem maior grau de dificuldade, ou, identificar quais os comportamentos errados mais frequentes dos alunos em cada tipo de problema;

Performance On Curriculum: Esse componente faz um processamento nos dados relacionados à evolução do aprendizado dos estudantes em um nível maior de granularidade que o componente *Performance On Curriculum*, ou seja, a nível de componentes do domínio, os currículos. Isso faz importante no sentido de obter quais os principais problemas no aprendizado de cada aluno ou em cada grupo de aluno. Por exemplo, pode-se recuperar quais os currículos que um determinado grupo de alunos obteve a menor nota ou qual as estratégias pedagógicas que possuem o maior rendimento por currículo;

Knowledge Level Report: Esse componente avalia o conhecimento do estudante de forma instantânea, de tal forma que um estudante, ou qual grupo de estudantes possuem um nível específico de conhecimento. Isso se faz importante para saber quais alunos, ou qual grupo de alunos, possuem um nível de conhecimento específico ou como está o nível de conhecimento de um grupo de alunos com um determinado conjunto de características. Cada um desses componentes provém uma interface distinta (*PerformanceOnDomain-I*,*HistoryOfResolutions-I*,*PerformanceOnCurriculum-I* e *KnowledgeLevelReport-I*) para que o componente *Manager Of Report* possa fazer solicitações e distribuir os resultados nos componentes da camada User Session.

4.5.4 Registro

Para que possam ser utilizados os componentes de forma adequada, faz-se necessário que o estudante passe por um processo de registro que permita que todo o sistema saiba exatamente qual aluno está interagindo com ele, e dessa forma possa adequar suas ações de forma

personalizada. Os componentes atribuídos com essa responsabilidade são:

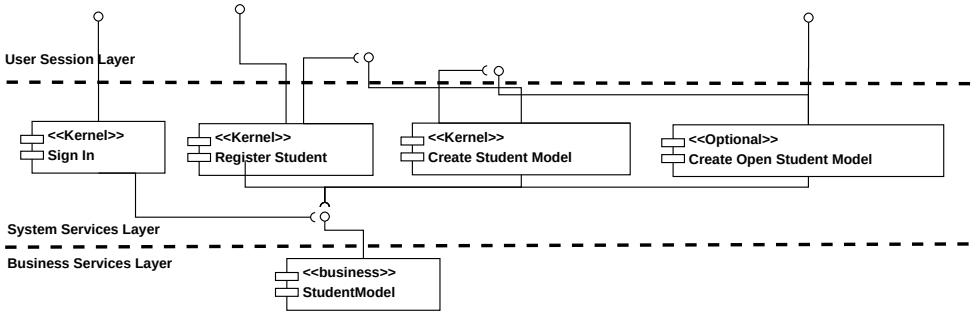


Figura 4.25: Artefatos de Registro no Sistema Tutor

Sign In: Este componente é responsável por receber solicitações de *login*, e responder com a identificação temporária para que os componentes da camada User Session possa se utilizar dos outros serviços da camada *System Service*;

Register Student: Após realizado o *Sign In*, o estudante poderá interagir com os demais componentes de forma segura e identificada. Nesse sentido, no momento que um determinado estudante estiver fazendo o seu *login*, o sistema irá verificar se já existe um registro atribuído a ele, caso contrário, só o componente de registro estará disponível para esse suposto estudante. Esse componente é responsável por submeter um questionário contendo informações básicas de um estudante (como endereço, idade, cidade e nível de escolaridade). Só depois desta etapa realizada é que é permitido ao estudante interagir com os recursos educacionais;

Create Student Model: Da mesma forma que o estudante só poderá interagir com os recursos do sistemas depois que o seu registro ter sido criado, faz-se necessário que o aluno tenha o seu modelo de estudante criado pelo sistema. O componente *Create Student Model* tem essa responsabilidade de criar o modelo do estudante preliminar para que o sistema tenha um prévio conhecimento sobre o estudante. Assim, com esse propósito, o sistema submete ao aluno uma prova com o objetivo de identificar o nível de conhecimento do aluno no domínio de conhecimento que o aluno deseja se submeter a aprender. Além dessa prova, esse componente solicita informações sobre o objetivo do

estudante em relação ao domínio, de forma que ele possa ajustar suas interações posteriores. Essa solicitação se dá através do componente *AssessmentResource* (detalhado na Subseção 4.5.5).

Create Open Student Model: O Componente *Create Open Student Model* cria a possibilidade ao estudante de monitorar todo o progresso e evolução da sua aquisição de conhecimento. Além disso, o componente oferece mecanismos para que os componentes da camada *User Session* possam prover ao aluno mecanismos para que o mesmo possa discutir e questionar sobre o modelo do estudante que o sistema esboçou. Assim, através dessas responsabilidades, o componente provê duas interfaces, uma interface para que o componente *Create Student Model* possa instanciar o modelo aberto do estudante, no momento que o modelo convencional for criado; e outra interface para que os componentes da camada User Session possam se utilizar dos serviços oferecidos por um modelo aberto do estudante.

4.5.5 Recursos de Avaliação

Uma das etapas do processo de criação do modelo do estudante é a avaliação inicial do conhecimento do estudante no domínio que o mesmo escolheu a aprender, conforme dito na Subseção anterior. Esse procedimento é feito, de fato, através do componente *AssessmentResource*, que se utiliza obrigatoriamente do componente *Test* para criar um teste adequado ao estudante, e, opcionalmente, para submeter ao estudante um formulário mais elaborado no sentido de obter informações de alto nível sobre o interesse do estudante no domínio.

Esse formulário elaborado pode ser utilizado por pessoas que buscam um STI de forma autônoma, normalmente por pessoas com um nível de maturidade maior, que já sabem exatamente o que quer. Portanto, se o tutor for utilizado por pessoas com um nível baixo de maturidade, como crianças, esse formulário pode não ser pertinente.

4.5.6 Resolução de Problemas

Um dos passos importantes no processo de ensino é a resolução do problema. Esse passo é importante porque cria a possibilidade de oferecer ao aluno uma visão de como o problema deve ser respondido. Na linha de produto proposta nesta tese, quando um componente da

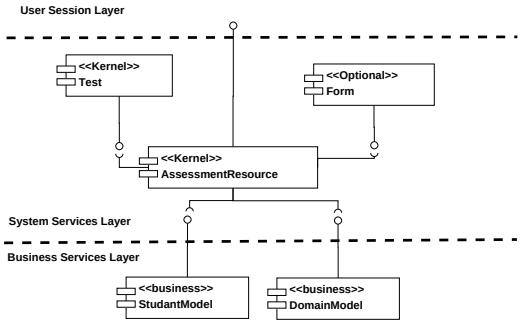


Figura 4.26: Artefatos relacionados aos tipos de avaliação.

camada *User Session*, ou, o próprio componente *tutoring* precisa requisitar uma resolução de problema em específico, ele recorre ao componente *Manager of Problem Resolution*, que por sua vez, utiliza-se de um dos componentes descritos a seguir que possui especificamente e distintamente essa característica:

General Form Resolution: Esse componente é alternativo com o componente *Step by Step Resolution*, nesse sentido, esse componente oferece um mecanismo direto na resolução do problema. Assim, passos relacionados a outros currículos são abstraídos;

Step by Step Resolution: Por outro lado, o componente *Steo by Step Resolution* proporciona um mecanismo de resolução de problema Step by Step de forma que alunos iniciantes possam entender todos os conceitos envolvidos, inclusive aqueles relacionados a curriculos que formam, de alguma forma, pré-requisitos;

Resolver Of Problems: Esse componente dá uma resposta, com uma justificativa, sem demonstrar os passos em como chegar nela. Por outro lado, ele cria a possibilidade de oferecer a explicação da resolução do problema desenvolvida por outros estudantes.

4.5.7 Estratégias Pedagógicas

As estratégias pedagógicas são utilizadas para servir de consulta para o componente *Tutoring* verificar qual a próxima a ação a ser executada. Nesse sentido, o componente *PedagogicalStrategiesManager* processa para cada requisição recebida, qual a melhor estratégia a ser utilizada. Para isso, ele pode se utilizar de seis componentes (*Situated*, *Behaviorist*, *Socratic*, *Constructivist* and *Cognitive*).

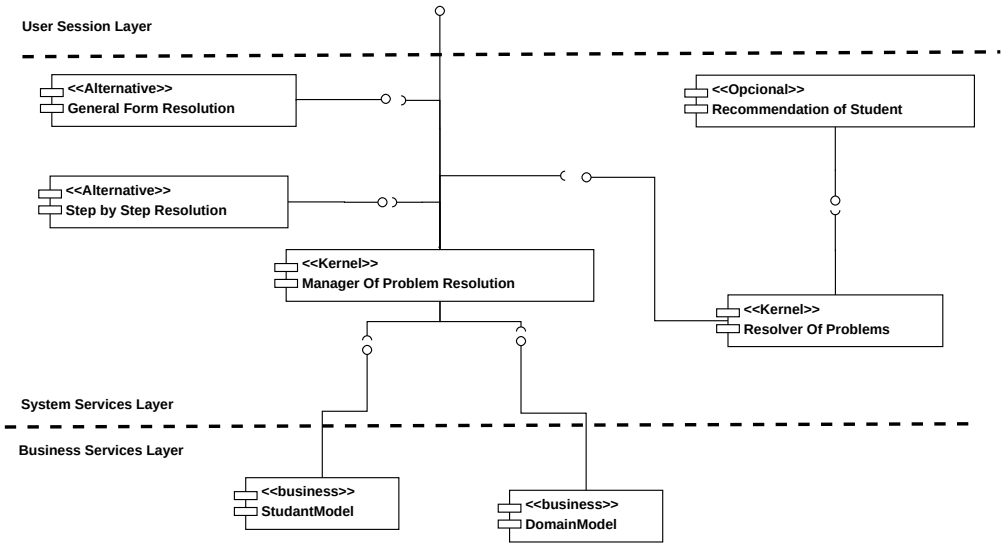


Figura 4.27: Artefatos relacionados a resolução de problemas

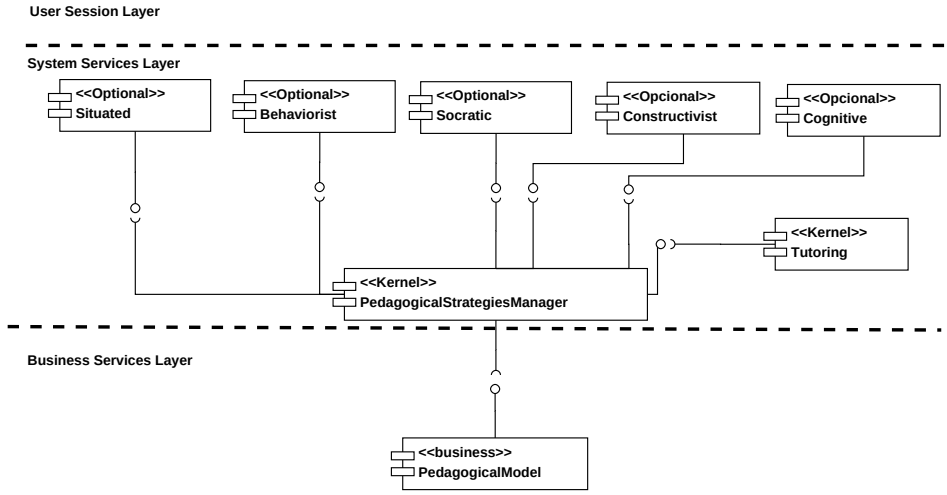


Figura 4.28: The Pedagogical Strategies Artefacts

4.6 Representação Semântica

Nesta seção encontram-se os modelos semânticos expressos em ontologias para representação de linhas de produtos de software juntamente com o seus respectivos domínios de produtos. Essa representação é importante para que se possa haver um processo automatizado e dinâmico no processo de construção, configuração de manutenção da linha de produto. Na Subseção 4.6.2 está descrita a representação semântica de linhas de produto, que é utilizada como uma meta-representação de linha de produto de software. Dessa forma, com esta representação, é possível descrever linhas de produtos com ontologias.

4.6.1 Ontologias Educacionais

As ontologias educacionais têm por objetivo descrever as características inerentes aos objetos de estudo. Elas são divididas em três módulos: i) modelo de domínio (o que ensinar), ii) modelo pedagógico (como ensinar) e iii) modelo do estudante (para quem ensinar).

O modelo de domínio (vide Figura 4.29) define os recursos educacionais e suas dependências através de uma visão multidimensional do conhecimento, baseado em uma revisão do modelo Mathema [27]. A classe *Domain* contem detalhes sobre os domínios nas quais se pretende ensinar. Esta classe é composta por Curricula, que utilizam o modelo Mathema para fazer o mapeamento das partições do domínio em uma estrutura curricular. As *PedagogicalUnits* modelam informações sobre os recursos necessários para abordar o Curriculum e os Resources modelam recursos educacionais que podem ser utilizados na plataforma.

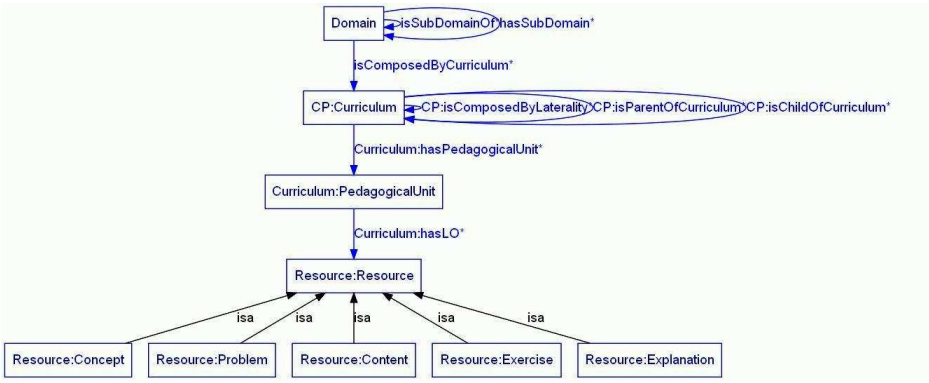


Figura 4.29: Ontologia de Domínio

O modelo pedagógico (vide Figura 4.30) é responsável por definir como uma interação pode ser conduzida [56] [38]. A partir de um *InstrutionalPlan* define-se a sequência de visualização dos recursos educacionais, podendo ser previamente especificada ou construída dinamicamente. Através de um *Sequencing* modela-se a sequência de recursos educacionais, que podem ser tanto *ResourceUnits*, como *ProblemUnits*. Os *ResourceUnits* representam recursos educacionais que não necessitam de avaliação para a tomada de decisão sobre o próximo recurso (e.g. conteúdo, exemplos, conceitos, entre outros). Já os *ProblemIUnits* definem recursos educacionais que necessitam de uma tomada de decisão sobre o próximo recurso (e.g. problemas). Caso o aluno não alcance o resultado esperado, um novo sequenciamento pode ser construído.

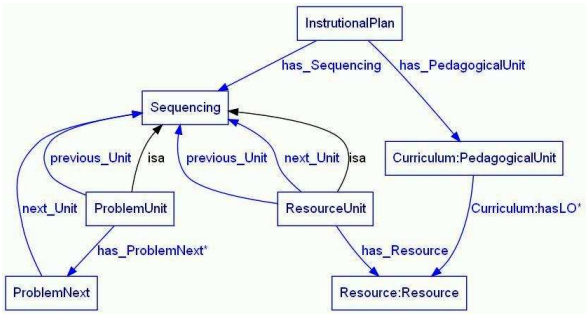


Figura 4.30: Ontologia Pedagógica

O modelo do estudante (Figura 4.31) é responsável por guardar informações, tanto estáticas como dinâmicas, do estudante [22] [23]. O User tem informações como nome, telefone, email, entre outras, além disso este deve definir quais seus objetivos de aprendizagem através do *LearningGoal*. *CognitiveInformation* são construídas dinamicamente para registrar as interações do estudante (InteractionLearnerData). Ressaltando-se que por meio do *AggregateData* tem-se informações como, média de tempo das respostas, tentativas para resolver um determinado problema, entre outras.

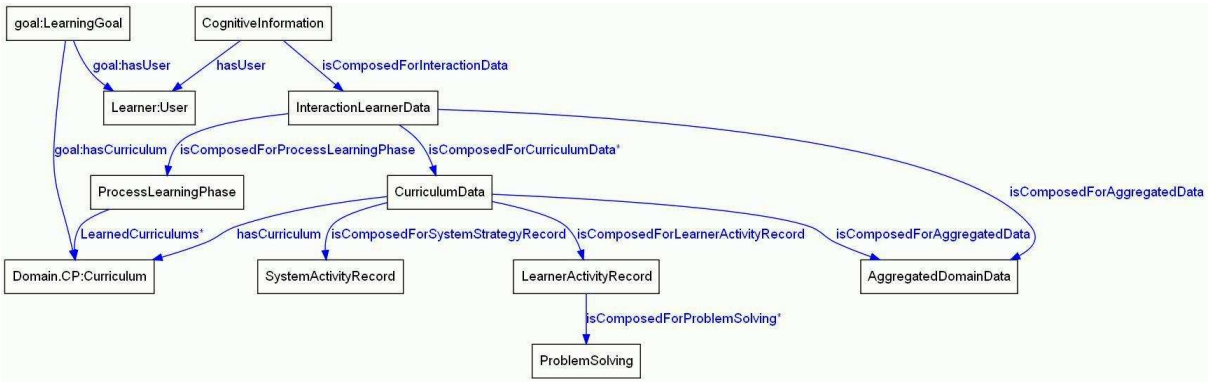


Figura 4.31: Ontologia do Estudante

4.6.2 Descrição Semântica de Linha de Produto de Software

Na Figura 4.32 está descrita a ontologia que representa uma meta-especificação de linha de produto de software. Nesta ontologia foram utilizadas quatro entidades/classes para fazer a representação inicial de uma linha de produto. A seguir está representada a descrição de cada uma dessas classes:

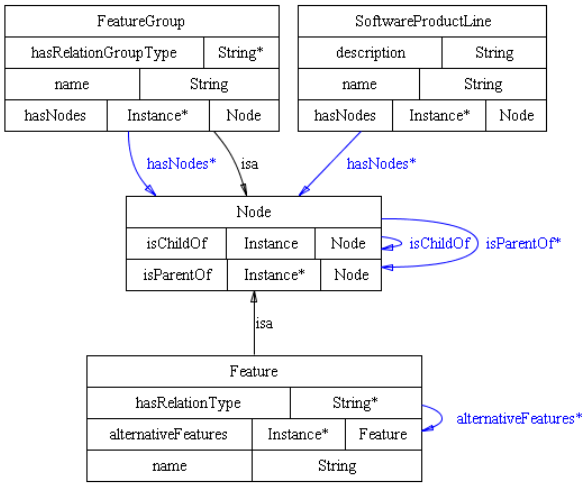


Figura 4.32: Ontologia para Representação de Linha de Produto de Software

SoftwareProductLine: Esta classe representa uma linha de produto de software. Ela possui como elementos primitivos o nome e a descrição. O nome representa a forma como ela é chamada, enquanto a descrição representa uma explicação do contexto que a linha de produto foi criada e como a mesma pode ser utilizada. Além disso, na representação proposta nesta tese, uma linha de produtos possui vários nós que são utilizados para representar a hierarquia de *features* que uma determinada linha de produto possui;

Node: Um *Node* tem a mesma representação gráfica que um nó em uma árvore. Dessa forma, o nó pode ter pai e filhos, conforme o local que o mesmo se encontre em uma estrutura. Por isso, esta classe tem as propriedades *isChildOf* e *isParentOf* para representar em que local cada nó se encontra. A propriedade *isChildOf* é utilizada para representar o pai do nó em questão, e a propriedade *isParentOf* é utilizada para representar os filhos deste mesmo nó.

Feature: Essa classe representa uma *feature* de linha de produto. Assim, esta classe é uma especialização da classe *Node* e é utilizada com o propósito de montar o *Feature Model*. Neste caso, esta especialização da classe *Node* possui três propriedades: A primeira é a *hasRelationType*, que representa que tipo de *feature* este representa, ou seja, se a *feature* é obrigatória, alternativa ou opcional; A segunda propriedade é a *alternativeFeatures*, que é utilizada para representar com que *features* a feature em questão forma um conjunto de features alternativas, de tal forma que estas sejam al-

ternativas entre si; por fim, a terceira e última propriedade é a *name*, que representa o nome que a mesma é identificada;

FeatureGroup: Essa classe representa grupos de features que podem formar grupos que possuem um interpretação particular. Assim, é possível especificar (i) grupos indeterminados de features que só podem ser escolhidas 0 ou 1 *feature (zero-or-one feature group)*; (ii) grupos de *features* intederminados que devem ser escolhidos ao menos uma *feature (at-least-one feature group)*; (iii) grupos de *features* indeterminados que só pode ser escolhida exatamente uma *feature (exactly-one feature group)*. (iv) Um grupo de *features* que pode ser escolhida qualquer quantidade de *features (zero-or-one feature group)*.

Além disso, algumas regras são importantes para que uma verificação automatizada possa ser realizada. Além de possibilidade que novas restrições possam ser adicionadas neste processo. Primeiramente, as regras da Figura 4.33 representam as regras que são disparadas no momento que uma linha de produto é especificada.

Rule 1: Selecionar todas as features, para que se possa ter um conjunto delas em um lugar na memória;

Rule 2: Seleciona as *features* que sejam, ao mesmo tempo, obrigatória e alternativa. Isto não pode acontecer, e por isso, deve haver alguma regra que faça esta verificação no momento de uma especificação de linha de produto;

Rule 3: Seleciona as regras que são opcionais e alternativas a uma outra ao mesmo tempo. Isto também não pode acontecer, e por isso, deve haver algum regra que faça esta verificação no momento de uma especificação de linha de produto;

Rule 4: selecionar as alternativas, para que haja um conjunto desta features na memória;

Rule 5: Buscar *features* que são alternativas e que possuem a propriedade *alternativeFeatures* não especificadas. Dessa forma, é possível combinar com a regra anterior quais as features que são alternativas, mas, no entanto, não foi especificado quais o grupo de features alternativas a mesma. Isso é feito em conjunto a regra anterior;

```
**************************************************************************
Rule 1:
**************************************************************************
SPL:Feature(?f)-> sqwrl:select(?f)

**************************************************************************
Rule 2:
**************************************************************************
SPL:Feature(?f) ^ SPL:hasRelationType(?f,"Mandatory") ^
abox:hasProperty(?f,SPL:alternativeFeatures) -> sqwrl:select(?f)

**************************************************************************
Rule 3:
**************************************************************************
SPL:Feature(?f) ^ SPL:hasRelationType(?f,"Optional") ^
abox:hasProperty(?f,SPL:alternativeFeatures) -> sqwrl:select(?f)


**************************************************************************
Rule 4:
**************************************************************************
SPL:Feature(?f) ^ SPL:hasRelationType(?f,"alternative")
                              -> sqwrl:select(?f)


**************************************************************************
Rule 5:
**************************************************************************
SPL:Feature(?f) ^ SPL:hasRelationType(?f,"alternative") ^
abox:hasProperty(?f,SPL:alternativeFeatures) -> sqwrl:select(?f)


**************************************************************************
Rule 6:
**************************************************************************
SPL:Feature(?f) ^ abox:hasProperty(?f,SPL:name)
     -> sqwrl:select(?f)


**************************************************************************
Rule 7:
**************************************************************************
SPL:Feature(?f) ^ abox:hasProperty(?f,SPL:hasRelationType)
                              -> sqwrl:select(?f)
```

Figura 4.33: Regras para Representação de Linha de Produto de Software

Rule 6: Seleciona as *features* que têm um nome. Assim, combinando com a primeira regra, é possível identificar quais *features* não possuem um nome;

Rule 7: Selecionam as *features* que têm algum tipo de relação (obrigatória, alternativa ou opcional). Através desta informação é possível fazer uma combinação com a primeira regra para identificar quais *features* não possuem relação.

4.6.3 Descrição Semântica do Modelo de Decisão

Nesta seção estão descritos os elementos semânticos necessários para representar e restringir as decisões no processo de configuração de um produto. Primeiramente, a ontologia rep-

resentada na Figura 4.34 é uma extensão da ontologia da Figura 4.32, apresentada na Subseção 4.6.2. Conforme pode ser visto na Figura 4.34, ela possui informações pertinentes a um aprendiz, conforme a ontologia *ForBILE.Learner* [13], onde as primeiras são descritas a seguir:

LearnerIdentification: Representa informações sobre a identificação pessoal do aprendiz, tais como, nome, endereço, telefone, idade, e assim por diante;

LearnerAffiliation: Representa informações sobre a instituição que o aprendiz está vinculado;

LearnerRole: Representa o papel que o estudante está exercendo no processo de ensino aprendizagem.

Além disso, as regras da Figura 4.35 são utilizadas para fazer gerenciar as decisões referentes a escolha de um produto de uma linha de produto qualquer:

Rule A: Seleciona todas as *features* que são obrigatórias e que não estão selecionadas. Isso vai de encontro com uma regra elementar em uma linha de produto onde todas as *features* obrigatórias devem estar em todos os produtos;

Rule B: Analisa *features* pares (pai e filho), onde o pai esteja eliminado e o filho selecionado;

Rule C: Verifica se existem mais de uma *feature* selecionada em um conjunto de *features* alternativas;

Rule D: Seleciona todas as *features* alternativas e eliminadas, para ajudar nas duas regras seguintes;

Rule E: Para cada resultado da regra D, analisam-se todas as *features* alternativas ao resultado corrente;

Rule F: Para cada resultado da regra D, analisam-se todas as *features* alternativas ao resultado corrente, mas que estejam eliminadas. Dessa forma, comparando os resultados da Regra E, é possível identificar se existe algum conjunto de *features* alternativas que não possui nenhuma *feature* selecionada.

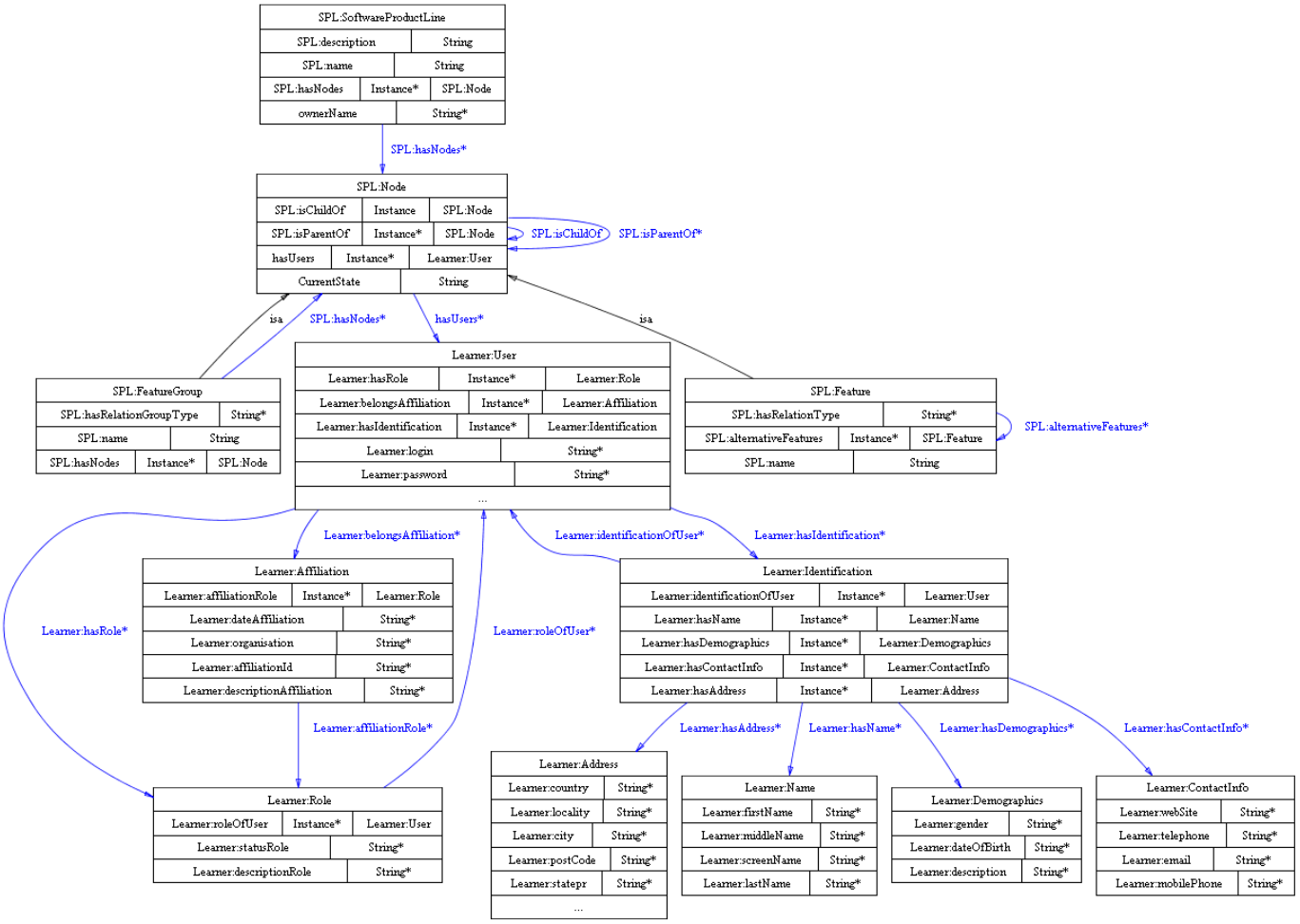


Figura 4.34: Ontologia para o Modelo de Decisão

```
**************************************************************************
Rule A:
**************************************************************************
SPL:Feature(?f) ^ SPL:hasRelationType(?f, "mandatory") ^
decisionModel:CurrentState(?f, "ELIMINATED")-> sqwrl:select(?f)
**************************************************************************
Rule B:
**************************************************************************
SPL:Feature(?p) ^
decisionModel:CurrentState(?p, "ELIMINATED") ^
SPL:isParentOf(?p, ?f) ^
decisionModel:CurrentState(?f, "SELECTED")
-> sqwrl:select(?f)

**************************************************************************
Rule C:
**************************************************************************
SPL:Feature(?f1) ^ SPL:Feature(?f2) ^
SPL:alternativeFeatures(?f1, ?f2) ^
decisionModel:CurrentState(?f1, "SELECTED") ^
 decisionModel:CurrentState(?f2, "SELECTED")
-> sqwrl:select(?f1)
**************************************************************************
Rule D:
**************************************************************************
SPL:Feature(?f) ^
decisionModel:CurrentState(?f, "ELIMINATED") ^
SPL:hasRelationType(?f, "alternative") -> sqwrl:select(?f)

**************************************************************************
Rule E:
**************************************************************************
SPL:alternativeFeatures('?f', ?z) -> sqwrl:select


**************************************************************************
Rule F:
**************************************************************************
SPL:alternativeFeatures('?z', ?x)^
decisionModel:CurrentState(?x, "ELIMINATED")
-> sqwrl:select(?x)
```

Figura 4.35: Regras para Validação de Produtos

4.7 Evolução Automatizada

Nesta seção está descrito o mecanismo de evolução de *features*. Assim, em tempo de execução, deve ser possível atualizá-las conforme as necessidades do aprendiz. Além disso, estes podem ser tanto descobertos com o propósito de oferecer uma funcionalidade mais adequada para o aprendiz, ou podem ser adicionados em tempo de execução no sistema. Assim, a ideia é que a linha de produto semântica dê suporte a três tipos de evolução:

Evolução *Top-down*: Esse tipo de evolução (vide Figura 4.36) ocorre quando uma feature já existente é modificada. Por conta disso, todos os produtos já existentes precisam ser atualizados da mesma maneira:

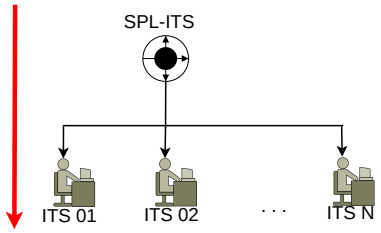


Figura 4.36: Evolução *Top-down*

Evolução *Botton-up*: Esse tipo de evolução (vide Figura 4.37) ocorre quando um produto adiciona uma nova *feature*. Através disso, outros produtos podem utilizar esta nova *feature*. Então, faz-se necessário que esta seja adicionada na plataforma, para que ela esteja disponível na construção de novos produtos, ou, para que produtos já existentes possam ser reconfigurados.

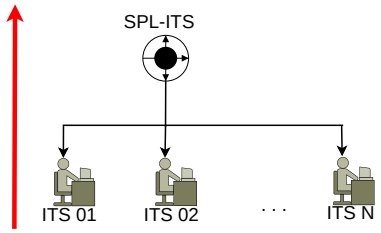


Figura 4.37: Evolução *Bottom-up*

Evolução através de *Semantic Web Services*: Esse tipo de evolução (vide Figura 4.38) ocorre quando uma nova *feature* é possível de ser composta a partir de serviços semân-

ticos. Isso é importante para que os diversos serviços possam ser aproveitados nos sistemas tutores inteligentes.

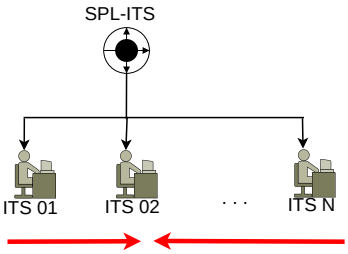


Figura 4.38: Evolução através dos *Semantic Web Services*

Para isto, além da descrição semântica dos artefatos, faz-se necessária a implementação de dois recursos que são apropriados para a evolução considerada anteriormente: Componente Semântico e o Meta Componente. Estes dois devem ser incorporados dentro de toda linha (vide Figura 4.39) descrito abaixo:

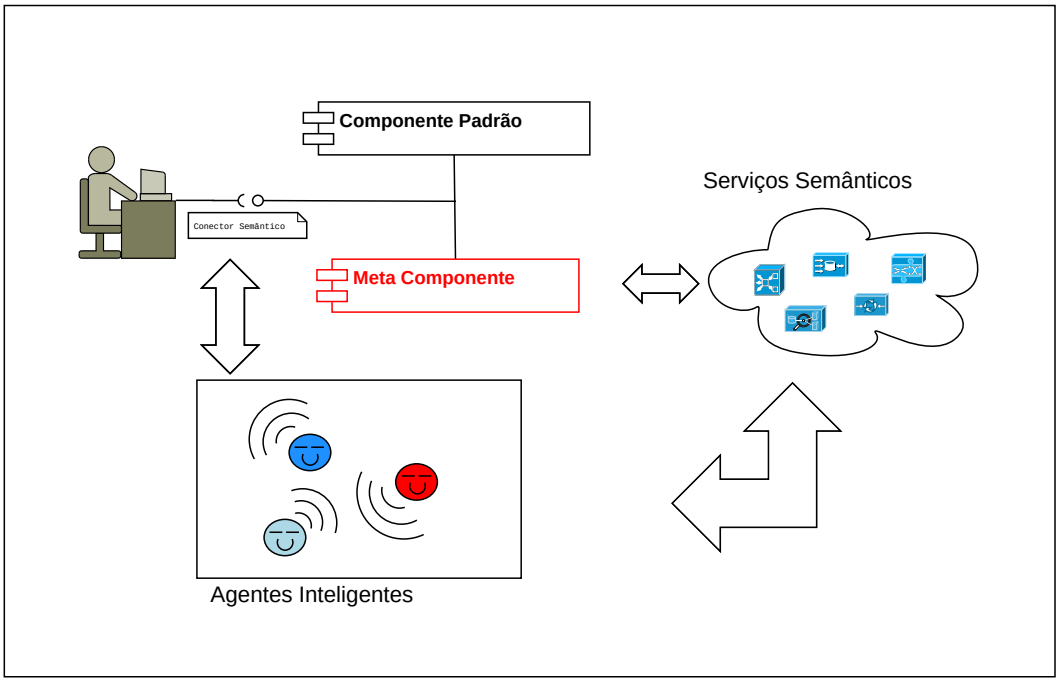


Figura 4.39: Arquitetura de Evolução

Componente Tutor: O Componente tutor é um componente que faz parte dos componentes que são estáticos dentro da arquitetura especificada. Além disso, este componente faz acesso a vários outros componentes para executar as suas tarefas, e neste caso,

por exemplo, inicialmente, o *componente Conector A*. A ideia é que este componente *Conector A* represente uma determinada *feature*. No entanto, este componente pode não ser o mais adequado para um determinado perfil de estudante, e por isso, pode ser necessário que o sistema tutor seja reconfigurado para que o componente seja substituído por um outro componente;

Conector Semântico: Representa um componente que foi configurado a priori para atender a um determinado conector, no entanto, o mesmo foi estabelecido como um componente possível de mudança automatizada;

Ontologia: Assim, para uma eventual mudança automatizada, faz-se necessário que algum mecanismo formal seja utilizado para que agentes de software possam analisar o momento que uma substituição deve ser feita, e além disso, analisar como esta mesma mudança deve ser conduzida;

Agentes de Descoberta e Composição de Serviços: Essas mudanças devem ser perceptíveis por agentes de software, e estes mesmo agentes devem identificar se existem novas possibilidades de componentes;

Serviços Semânticos: A construção de novos componentes acontece quando novos serviços podem ser utilizados para fornecer alguma funcionalidade variável já especificada no sistema. Isso é importante para que o padrão dos novos componentes seja conhecido pelos agentes computacionais de sentido de fazer com que os mesmos possam identificar com precisão os novos serviços;

Meta Componente: Este componente tem por função abstrair o acesso aos serviços semânticos por parte do conector semântico. Então, quando o conector semântico solicita um serviço composto por serviços semânticos, na realidade, uma requisição é enviada ao componente semântico.

4.7.1 Conector Semântico

Para concretização dessa arquitetura, foi necessário uma modelo de componentes. Onde o mesmo é responsável por definir um conjunto de padrões para implementação, documen-

tação e implantação dos componentes. Além disso, ele também tem como objetivo especificar como será o acesso as interfaces de cada componente.

Para implementação da modelagem proposta no Capítulo 4 foi utilizado o modelo de componentes *COSMOS*[41]. Ele foi desenvolvido de modo facilitar a evolução do sistema. Além disso, o acesso a todas as interfaces do componente é feito através da interface *IManager*. Desse modo, só é necessário instanciar uma interface para ter acesso as outras.

Pode-se observar que o conector é a entidade de software responsável por associar o kernel do sistema tutor inteligente com os componentes satélites. Além disso, o conector, através do algoritmo descrito abaixo pode chavear os componentes satélites, seguindo as preferências de cada usuário. Portanto, este é um elemento importante no que se refere à personalização de todos os ITS's. O processo de troca dos componentes satélites será exemplificado a partir do conector responsável pelo chaveamento entre os componentes satélites responsáveis dos problemas, ou seja, os componentes: Múltipla Escolha, Preencher Lacunas, Relacionar Colunas e Verdadeiro e Falso. O algoritmo a seguir mostra como ele decidirá qual componente será utilizado:

- *Passo 1*: Em um primeiro momento, o algoritmo busca pela ontologia específica do produto do usuário.
- *Passo 2*: No segundo passo, o algoritmo busca pela componente especificado para fornecer o serviço de responsabilidade do tutor. Assim, desta forma, é possível fornecer um mecanismo dinâmico de escolha de componentes. Assim, para que isto aconteça, faz-se necessário uma mudança na ontologia, onde a mesma pode ser utilizada como mapeamento.
- *Passo 3*: Assim, depois da implementação correta ter sido estabelecida, a requisição será delegada para o componente correto.

Nesse sentido, o conector seleciona entre as implementações possíveis, qual a que realmente deve ser utilizada. Mais especificamente, o pseudo-código abaixo mostra como foi implementado o algoritmo acima:

This way, the connector switches between the interface implementations, changing the manager of each component. Based on this algorithm was developed a pseudo-code below, where it was implemented in all the connectors of the architecture.

Require: Identifiers: current resource and user *URI*.

Interface interface;

IManager manager;

{interface and manager do not start with some implementation. Because they do not known a priori to which implementation is related to the student.}

$kindResource \longleftarrow queryGetKindResource$

if kindResource is equal to SatelliteComponent1 **then**

$manager \longleftarrow SatelliteComponent1.getManager()$

$interface \longleftarrow manager.getProvidedInterface(\text{“}kindResource\text{”})$

end if

if kindResource is equal to SatelliteComponent2 **then**

$manager \longleftarrow SatelliteComponent2.getManager()$

$interface \longleftarrow manager.getProvidedInterface(\text{“}kindResource\text{”})$

end if

(...)

return Identifier next resource

Essa lógica repete-se para todo componente satélite a qual o conector pode chavear. No final, o conector retorna a implementação da interface referente ao usuário, ou seja, se o usuário desejou receber problemas do tipo do Verdadeiro ou Falso, o conector retorna-rá a implementação da interface IProblem, presente na arquitetura (Figura **??**), através do componente satélite TrueOrFalse Problem. Como pode ser visto, apesar do código do conector ser relativamente curto, sua importância é bastante significativa. Pois é através deste que tornou-se possível viabilizar uma mudança dinâmica nos componentes.

4.7.2 Meta Componente

O Meta Componente visa proporcionar a composição e invocação automática de Serviços Semânticos de uma maneira extensível e também transparente para o usuário os sistemas tutores. No entanto, ele não se preocupa com a composição dos serviços, pois isto é feito pelo SMA. Além disso, o *OWL-S* foi escolhido para descrever os serviços semanticamente, assim como é baseada em *OWL*, que permite uma fácil integração de ferramentas e instrumentos

OWL-S propostos para a manipulação de ontologias em *OWL*, como Jena (Jena é um dos mais amplamente utilizadas APIs Java para *RDF* e *OWL*). Além disso, uma série de ferramentas têm sido propostos para a descoberta e composição dinâmica de serviços baseados em *OWL-S* [70],[89], [54],[18].

Este componente foi desenvolvido utilizando a linguagem Java. A API OWL-S [2] foi utilizada para manipulação de descrições semânticas dos serviços. A Figura 4.40 ilustra a arquitetura de alto nível. Os módulos desta arquitetura são comentados abaixo:

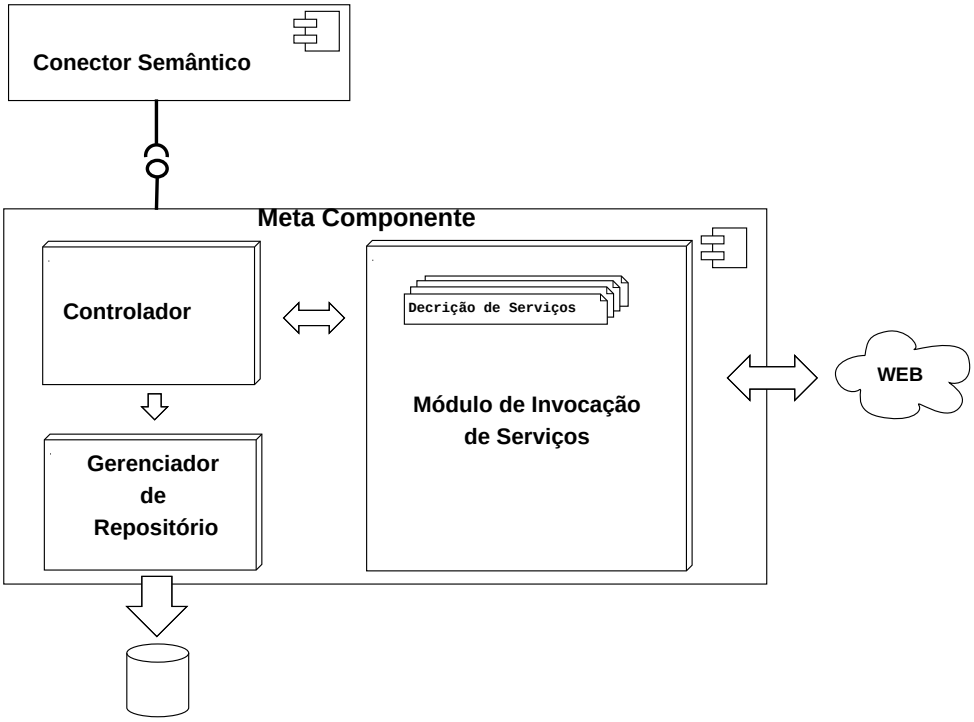


Figura 4.40: Arquitetura do Meta Componente

- ***Controlador***: Este módulo é responsável por descobrir qual é o melhor serviço em um repositório para atender as descrições enviadas por seu usuário, este módulo possui um conjunto de algoritmos de descoberta que efetuar essa tarefa;

- ***Módulo de Invocação de Serviços***: Este módulo é responsável por invocar automaticamente o serviço que foi fornecido pelo usuário ou descobertos por módulo. Este módulo deve ser capaz de invocar simples (um único serviço) e processos compostos de uma forma transparente;

[2] available at http://on.cs.unibas.ch/owls-api/

- ***Gerenciador de Repositório***: Este módulo tem a função de armazenar os serviços que são usados pelas descrições dos serviços. Ele utiliza o Sesame[17] para armazenar e manipular ontologias que descrevem os serviços. Como o sesame armazena ontologias em uma base de dados, este módulo possui rotinas para verificar atualizações que podem ocorrer nas ontologias previstas na Web e atualizar o conteúdo da base de dados. O *Gerenciador de Repositório* tem acesso ao ***Repositório***;

- **Controlador**: Este módulo é responsável para ajustar o sistema de acordo com as configurações feitas pelo usuário, ou seja, esse módulo carrega os arquivos de configuração que indica a descoberta de algoritmos que serão utilizados, a invocação e mecanismos de política do repositório.

Capítulo 5

Engenharia de Aplicação

Este capítulo tem por objetivo esclarecer como os modelos desenvolvidos e demonstrados no Capítulo 4 são utilizados no processo de construção de Sistemas Tutores Inteligentes (STIs). Estes são importantes para mapear os conceitos envolvidos tanto em Sistemas Tutores Inteligentes como em Linhas de Produto de Software. Assim, na Seção 5.1 é demonstrado como os modelos são relacionados e utilizados na construção de um Sistema Tutor Inteligente. Na Seção 5.2 encontra-se a modelagem, com ontologias, da linha de produto, e como os produtos são modelados. Por fim, na Seção 5.3.1, encontra-se descrito uma ferramenta para proporcionar uma instanciação facilitada e correta dos sistemas tutores inteligentes. As telas da implementação de ferramentas propostos neste capítulo, encontra-se no Capítulo 6.

5.1 Modelagem de Sistemas Tutores Inteligentes

O Processo de modelagem de um Sistema Tutor Inteligente está relacionado aos modelos específicos de características gerais de Sistemas Tutores Inteligentes combinado com os modelos necessários para especificação de Linhas de Produto de Software. Dessa forma, quatro modelos foram combinados e compartilhados para que o domínio de Sistemas Tutores Inteligentes fosse adaptado e projetado no sentido de fornecer uma Linha de Produto de Software. Estes modelos foram reproduzidos em ontologias a fim de viabilizar o compartilhamento de conhecimento entre todos eles, viabilizando uma distribuição de forma organizada entre os mesmos, de tal forma que possa haver uma manutenção ou expansão

destes de forma desacoplada entre os demais modelos. Além disso, descrevê-los em ontologias permite que algoritmos de software sejam desenvolvidos para suportar todo o processo de construção dos produtos de forma precisa, dado que o significado de cada termo é anotado de uma forma não ambígua. Os modelos anotados e utilizados no processo de construção dos produtos são os seguintes:

LPS Ontology: Esta ontologia foi descrita na Seção 4.6.2. Ela foi projetada para ser utilizado como um meta-modelo para uma especificação semântica de *Feature Oriented Domain Analysis* (FODA) [51]. A partir deste modelo, é possível modelar ontologias de qualquer domínio de aplicação, dentro das especificações do próprio modelo. Assim, com este modelo separado dos demais, foi possível separar as regras específicas de modelagem de linha de produto, que são independentes do domínio que o mesmo é utilizado. Estas regras também foram definidas na Seção 4.6.2;

LPS-ITS Ontology: Esta especificação segue às definições da Seção 4.3. Isso é importante porque fornece de forma isolada todas as possibilidades de um sistema tutor, possibilitando que mudanças possam ser feitas de forma independente dos outros modelos, e independente dos domínios de conhecimento do tutor;

Forbile.Learner Ontology: Esta ontologia disponibiliza um mecanismo para representação de conhecimento de aprendizes. Assim, este formalismo pode ser compartilhado com outros ambientes ou ferramentas no sentido que funcionalidades possam ser integradas ou adaptadas. Ele é adaptada do trabalho [13];

Decision Model Ontology: Esta ontologia representa a instanciação propriamente dita de um Sistema Tutor Inteligente. Ela foi definida na Seção 4.6.3. Com esta ontologia é possível definir as especificidades de cada produto de forma adequada. Isso acontece porque existem regras específicas para este modelo, de tal forma que elas sejam são úteis para a verificação da instanciação dos produtos.

5.2 Representação Semântica das *Features* e dos Produtos

Nesta modelagem estão descritas todas as *features* juntamente com a definição do local, em relação à hierarquia onde a mesma se encontra. Nesse sentido, é importante entender o

significado das seguintes propriedades:

isChildOf: Esta propriedade serve para relacionar a *feature* pai da *feature* em questão;

isParentOf: Esta propriedade serve para relacionar a(s) *feature(s)* filha(s) da *feature* em questão;

hasRelationType: Esta propriedade serve para relacionar o tipo de *feature* em relação a sua possibilidade de existência nos produtos tendo apenas três tipos: *Alternative*, *Optional* e *Mandatory*;

name: Esta propriedade se refere ao nome da *Feature*;

alternativeFeatures: Esta propriedade é utilizada quando a *feature* é caracterizada como *alternativa*. Ele relaciona as *Features* que são alternativas a *feature* em questão.

No código abaixo se encontram três exemplos de *features* juntamente com suas respectivas definições, conforme as propriedades que foram detalhadas anteriormente.

Feature(Geral)

---

alternativeFeatures(Geral, Passo_a_Passo)
isChildOf(Geral, ExplicacaoResolucaoProblema)
hasRelationType(Geral, alternative)
name(Geral, Geral)

Feature(Aberto)

---

isChildOf(Aberto, ModeloEstudante)
name(Aberto, Aberto)
hasRelationType(Aberto, optional)

Feature(Avaliacao)

---

isParentOf(Avaliacao, Somativa)
name(Avaliacao, Avaliação)

hasRelationType(Avaliacao, mandatory)

No código acima se encontra a definição de três *features*. Segue a explicação destas:

Geral: É uma *feature* que é alternativa. Além disso ela é filha da *feature ExplicacaoResolucaoProblema*. Por fim, ela é uma *feature* alternativa da *feature* passo-a-passo;

Aberto: É uma *feature* filho da *feature* modelo do estudante. Além disso é uma feature opcional;

Avaliação: É uma *feature* que tem a *feature* Somativa como filha. Além disso, é uma *feature* obrigatória.

No processo de Instanciação de um produto, a exemplo de um Sistema Tutor Inteligente, faz-se necessário especificar para cada *feature* o estado corrente (propriedade *CurrentState*) de cada uma das *features*. A definição é realizada utilizando a ferramenta de autoria especificada na Seção 5.3.1. Para isto, para cada *feature* é necessário definir o seu *status* como *ELIMINATED* ou *SELECTED*. Além disso, a validação da escolha do estado corrente é feita utilizando-se as regras da Seção 4.6.3.

CurrentState(Aberto, ELIMINATED)
CurrentState(Avaliacao, SELECTED)
CurrentState(Cadastro, SELECTED)
CurrentState(Cognitiva, ELIMINATED)
CurrentState(Comportamentalista, ELIMINATED)
CurrentState(Construtivista, ELIMINATED)
CurrentState(Conteudo, SELECTED)
CurrentState(Curriculum, SELECTED)
CurrentState(DesempenhoCurriculum, ELIMINATED)
CurrentState(DesempenhoDominio, ELIMINATED)
CurrentState(Dica, ELIMINATED)
CurrentState(EstrategiaBaseadaTeoriaAprendizagem, ELIMINATED)
CurrentState(Estudante, ELIMINATED)
CurrentState(Exemplo, ELIMINATED)
CurrentState(ExplicacaoResolucaoProblema, SELECTED)
CurrentState(Formulario, ELIMINATED)
CurrentState(Geral, SELECTED)
CurrentState(HistoricoResolucoes, ELIMINATED)
CurrentState(Login, SELECTED)
CurrentState(ModeloDominio, SELECTED)
CurrentState(ModeloEstudante, SELECTED)
CurrentState(ModeloPedagogicoBaseadoProblema, SELECTED)
CurrentState(NivelConhecimento, ELIMINATED)
CurrentState(Passo_a_Passo, ELIMINATED)
CurrentState(Problema, SELECTED)
CurrentState(ProblemaMultiplaEscolha, SELECTED)
CurrentState(ProblemaPreencherLacunas, ELIMINATED)
CurrentState(ProblemaRelacionarColunas, ELIMINATED)
CurrentState(ProblemaVerdadeiroFalso, ELIMINATED)
CurrentState(Professor, ELIMINATED)
CurrentState(Prova, SELECTED)

CurrentState(Recomendar_par, ELIMINATED)
CurrentState(Recurso, SELECTED)
CurrentState(RelatorioAprendizagem, ELIMINATED)
CurrentState(Situada, ELIMINATED)
CurrentState(Socratica, ELIMINATED)
CurrentState(Somativa, SELECTED)
CurrentState(Tatica, ELIMINATED)
CurrentState(Tutor, ELIMINATED)
User(UsuarioPadrao)
rate(UsuarioPadrao, 0.0)
login(UsuarioPadrao, UsuarioPadrao)
password(UsuarioPadrao, senha)
numberOfAccesses(UsuarioPadrao, 0)
experienceLevel(UsuarioPadrao, Low)
CurrentState(VisualizacaoConteudo, SELECTED)
CurrentState(VisualizacaoDica, ELIMINATED)
CurrentState(VisualizacaoExemplo, ELIMINATED)
CurrentState(VisualizacaoMultiplaEscolha, SELECTED)
CurrentState(VisualizacaoPreencherLacunas, ELIMINATED)
CurrentState(VisualizacaoProblema, SELECTED)
CurrentState(VisualizacaoRecurso, SELECTED)
CurrentState(VisualizacaoRelacionarColunas, ELIMINATED)
CurrentState(VisualizacaoVerdadeiroFalso, ELIMINATED)
CurrentState(Wiki, ELIMINATED)

5.3 Ferramenta de Autoria

Este seção descreve uma ferramenta de autoria para Linha de produtos de software (LPS). Esse sistema se diferencia dos sistemas de autoria convencionais principalmente por utilizar ontologias para representar as LPS e seus produtos, o que permite personalizar, validar, implantar e gerenciar os produtos gerados de forma inteligente e flexível.

5.3.1 Arquitetura da Ferramenta de Autoria

A arquitetura da ferramenta de autoria baseia-se numa Arquitetura Heterogenia que combina os estilos arquiteturais *Pipes and Filters*, Centrado em Conhecimento, Componentes Independentes e Camadas. Porém, como pode ser visto na Figura 5.4, o estilo arquitetural utilizado como arquitetura de referência foi o estilo Centrado em Conhecimento. A representação lógica de alto nível da arquitetura (Figura 5.4) evidencia os três módulos principais do sistema: (1) *User Registration*, que realiza o cadastro dos usuários no sistema; (2) *SPL*

Registration que realiza o cadastro das *LPSs* e suas *URIs* e (3) *SPL Instanciation* que permite a criação de um produto. A partir da ontologia de uma LPS existente, relaciona as *features* selecionadas com seus componentes e o implanta no servidor WEB. O estilo arquitetural Centrado em Conhecimento, nesse caso, foi utilizado para facilitar a integração entre os módulos, tendo em vista o caráter incremental do desenvolvimento. Nessa arquitetura, cada componente implementa parte das funcionalidades e a comunicação entre eles ocorre através do compartilhamento de um mesmo repositório de dados.

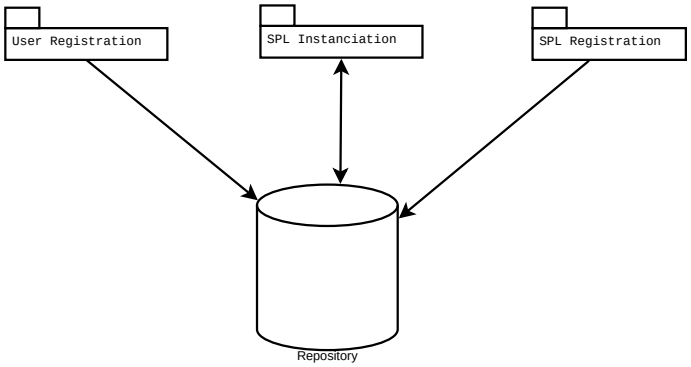


Figura 5.1: Arquitetura

No diagrama apresentado na Figura 5.4, é mostrada a arquitetura do S-AUTOLINES. Seus componentes arquiteturais são melhores descritos a seguir:

Repository - é o repositório de dados do sistema, responsável por prover uma interface comum e de alto nível aos outros componentes do sistema para manipular os SGBDs, sendo um deles relacional e outro baseado em ontologias. Esse componente representa a principal fonte de integração entre os demais componentes da arquitetura, que ocorre através do compartilhamento de dados.

User Registration - é responsável pelo cadastro dos usuários que poderão utilizar o aplicativo. Apesar de não fazer parte diretamente do fluxo de criação ou instanciação de LPSs, este componente compartilha o mesmo repositório de conhecimento (*Repository*).

SPL Registration - é responsável pelo cadastro das LPSs que poderão ser utilizadas para instanciar produtos. Apesar de não fazer parte diretamente do fluxo de criação de

usuário ou instanciação de LPSs, este componente compartilha o mesmo repositório de conhecimento (Repository).

SPL Instanciation - é responsável pela criação de produtos, a partir de uma LPS. Apesar de não fazer parte do fluxo criação de usuário e de LPSs, este componente compartilha o mesmo repositório de dados (Repository). Vale ressaltar que as LPSs que podem ser instanciadas já devem ter sido cadastradas previamente pelo componente LPS Registration. O componente SPL Instanciation é o foco principal deste trabalho; .

5.3.2 Detalhamento Interno do Componente *SPL Instanciation*

O *SPL Instanciation*, é responsável pela criação de produtos a partir de uma LPS, para tanto implementa componentes responsáveis pela autenticação do usuário no sistema, seleção da LPS a ser instanciada, personalização, validação, geração e implantação do produto. A arquitetura composta por seis componentes que compartilham o mesmo repositório de dados (*Repository*). Nessa arquitetura, cada componente implementa parte do processamento, que deve ser executado sequencialmente, onde a saída de um é a entrada do próximo componente; porém, a comunicação entre eles ocorre através do acesso a um mesmo repositório de dados.

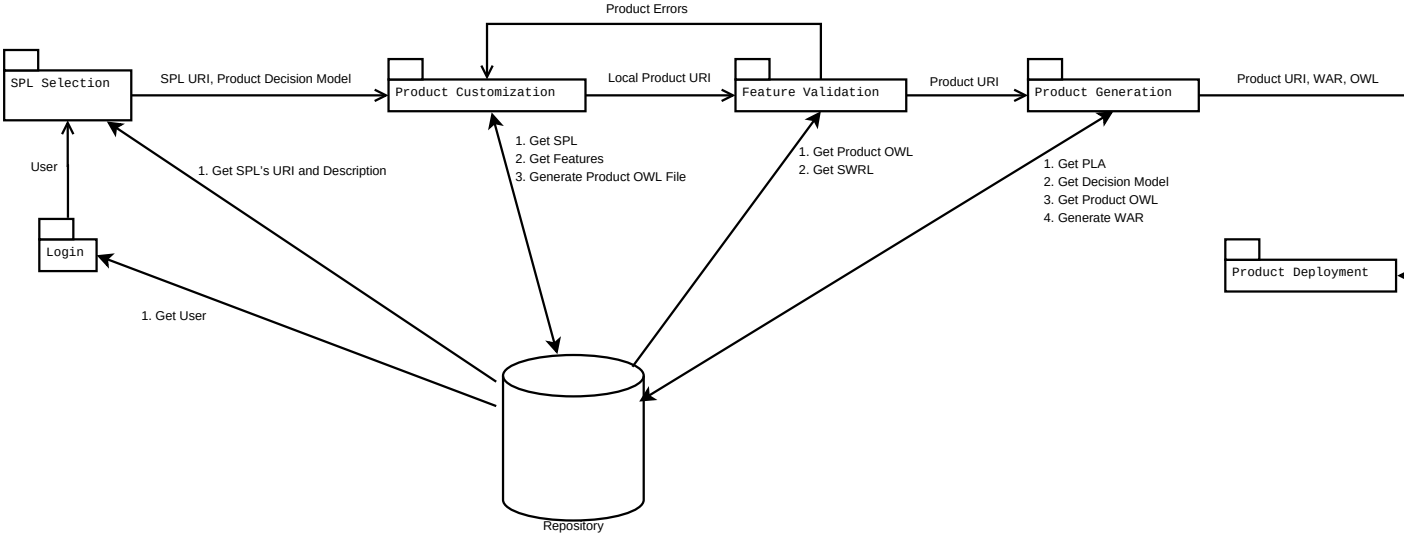


Figura 5.2: Detalhamento Interno do Componente *SPL Instanciation*

No diagrama apresentado na Figura 5.2, é mostrada a arquitetura do SPL Instanciation.

Seus componentes arquiteturais são melhor descritos a seguir:

Login - é responsável pela autenticação do usuário no sistema e representa o início do fluxo de execução. Os dados de *login* são recuperados a partir do componente *Repository*. Após corretamente autenticado no sistema, o usuário é direcionado para o componente *SPL Selection*;

SPL Selection - é responsável por obter as LPSs disponíveis no repositório de dados (*Repository*) e disponibilizá-las na *GUI* para que o usuário possa escolher qual LPS será instanciada. Após selecionar a LPS e informar o nome do produto a ser gerado, são passadas ao componente *Product Customization* as URIs da LPS e do *Product Decision Model*;

Product Customization - é responsável por instanciar a LPS, e suas *features* a partir do repositório de dados (*Repository*) para disponibilizar o *Feature Model* da linha na *GUI*, alterar e validar axiomaticamente as features selecionadas pelo usuário (*Product Decision Model*). Este componente possui um papel fundamental na criação dos produtos, uma vez que é responsável pela geração conceitual do produto e por consequência o seu armazenamento no repositório (*Repository*) na forma de ontologia do produto. Quando corretamente gerada e validada a ontologia, *Product Customization* chama o componente *Feature Validation* passando a URI do arquivo OWL gerado no servidor;

Feature Validation - é responsável por fazer a validação complexa do produto verificando as dependências e outros requisitos das features. Para validar o produto, utiliza uma engine para regras SWRLs sobre o arquivo OWL gerado, o que dá flexibilidade para alterar as regras de validação. Caso exista erros, retorna ao *Product Customization* passando os erros encontrados para que sejam corrigidos. Caso contrário chama o *Product Generation* passando a URI da ontologia do produto;

Product Generation - é responsável por gerar o produto a partir da sua ontologia e do *Decision Model*, instanciar os componentes referentes às *features* selecionadas e gerar ao final um arquivo WAR (*Web Archive*) pronto para ser implantado no servidor web. O WAR juntamente com o OWL do produto são passados para o *Product Deployment*;

Product Deployment - é responsável por implantar o produto (WAR e OWL) no servidor web e disponibilizar a URL do produto ao usuário.

5.3.3 Detalhamento Interno do Componente *Product Customization*

O *SPL Customization*, como já foi citado na Seção 5.3.2, é responsável por instanciar um produto da LPS, a partir de suas features. Sendo assim, a partir do repositório de dados (*Repository*), é necessário disponibilizar o *Feature Model* da linha na *GUI*, gerar o arquivo OWL do produto, alterar e validar axiomaticamente as features selecionadas pelo usuário (*Product Decision Model*). Sua Arquitetura interna baseia-se numa Arquitetura Heterogenia que combina os estilos arquiteturais Pipes and Filters e Centrado em Dados. Sendo composta por dois componentes: *Feature Selection* and *Axiomatic Validation* e *Product OWL Generation* ambos compartilham o mesmo repositório de dados (*Repository*).

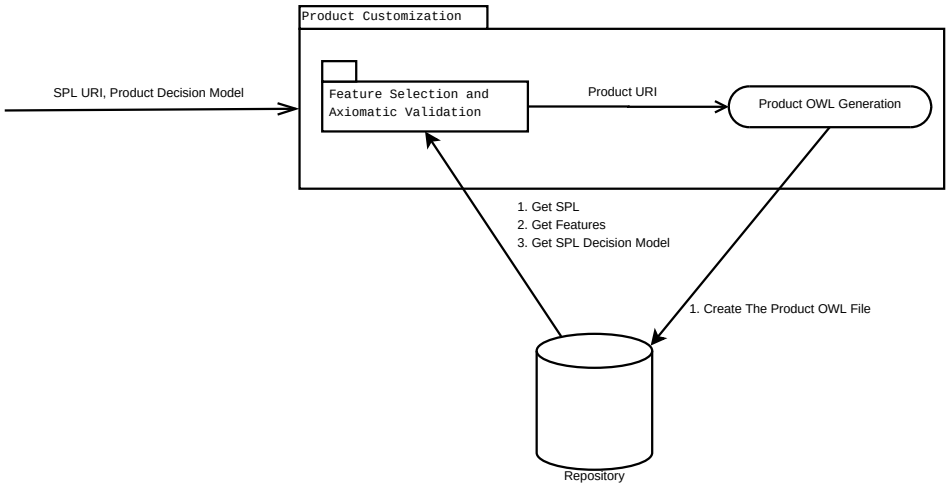
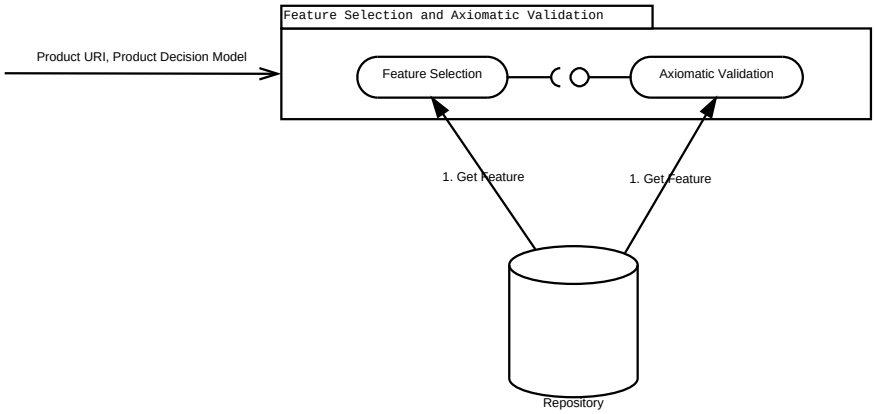


Figura 5.3: Detalhamento Interno do Componente *Product Customization*

No diagrama apresentado na Figura 5.3, é mostrada a arquitetura do *Product Customization*. Seus componentes são melhor descritos a seguir:

Feature Selection and Axiomatic Validation - é responsável por alterar o *Product Decision Model* de acordo com as *features* selecionadas pelo usuário e realizar a validação axiomática (regras imutáveis) do produto, informando as inconsistências para que sejam solucionadas pelo usuário. Sua Arquitetura interna baseia-se no estilo arquitetural de Componentes Independentes, implementando 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* precisa ser notificado para verificar a integridade do conjunto selecionado, de acordo com o *Decision Model* da LPS.



No diagrama apresentado na Figura 5.3.3, é mostrada 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 o repositório de dados (*Repository*), obter a LPS de acordo com o URI passada, obter suas *features* e alterar o *Product Decision Model* de acordo com as *features* selecionadas pelo usuário;

Axiomatic Validation - é responsável por validar axiomaticamente o *Product Decision 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 atualizar ou gerar o OWL do produto de acordo com a ontologia do banco;

Product OWL Generation - é responsável por atualizar o OWL existente ou gerar um novo OWL e persistir a ontologia do produto no banco utilizando o *Repository*. 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 complexas.

5.3.4 *Repository*

O *Repository*, como já foi citado na Seção 5.3.1, é o repositório de dados do sistema, responsável por prover uma interface comum e de alto nível aos outros componentes do sistema para manipular os SGBDs. Esse componente representa a principal fonte de integração entre os demais componentes da arquitetura, que ocorre através do compartilhamento de dados. Sua Arquitetura interna baseia-se numa Arquitetura em três Camadas.

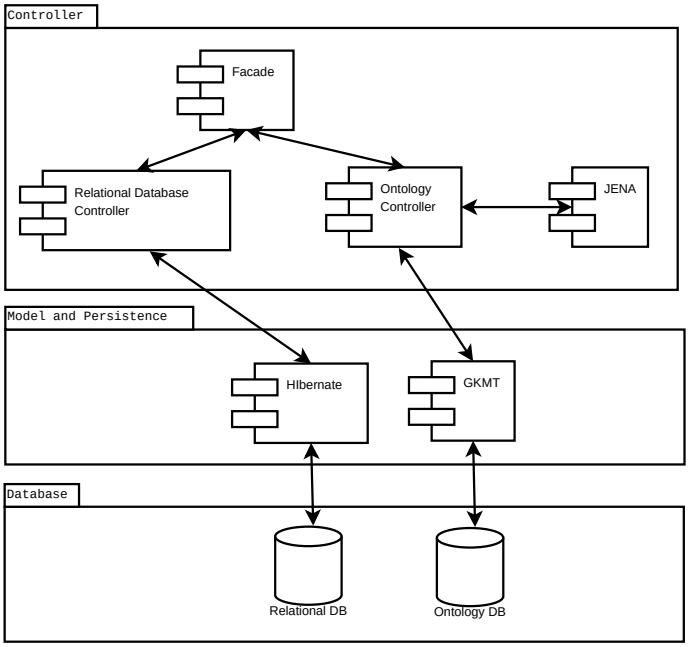


Figura 5.4: *Repository*

Controller - essa camada é responsável por gerenciar e repassar as requisições de manipulação do repositório de dados. Essa camada é composta pelos quatro componentes descritos a seguir:

- *Facade* - interface única para manipulação do *Repository* pelos componentes externos.
- *Relational Database Controller* - este componente é responsável por gerenciar e repassar as requisições de manipulação dos registros do banco relacional à camada *Model and Persistence*.
- *Ontology Controller* - este componente é responsável por gerenciar e repassar as

requisições de manipulação das ontologias à camada *Model and Persistence* e ao componete Jena.

- *Jena* - este componente é um framework open source em java para criação de aplicações semânticas. Provê um ambiente de programação para RDF, RDFS e OWL, SPARQL e inclui um motor de inferência baseado em regras.

Model and Persistence - essa camada é composta pelos componentes responsáveis pela manipulação dos bancos de dados existente na camada Database provendo um maior nível de abstração e flexibilidade em relação aos SGBDs utilizados. Essa camada é composta por dois componentes descritos abaixo:

- *Hibernate* - este componente é um framework de persistência em java open source que realiza o mapeamento objeto-relacional entre os objetos da camada Controller e o banco relacional (Relational DB) e a persistência dos objetos no banco.
- *GKMT* - este componente é um framework desenvolvido pelo GrOW[1] para manipular as ontologias no *Ontology DB* e obter os seus respectivos objetos.

Database - essa camada se limita ao fornecimento de dados via SGBDs, sem qualquer lógica de negócio. Sua arquitetura possui apenas dois componentes: *Relational DB* e *Ontology DB* descritos abaixo:

- *Relational DB* - este componente é um banco de dados relacional.
- *Ontology DB* - este componente é um banco de dados para ontologias. Nos testes será utilizado o Sesame que é um framework open source para inferência, armazenamento e consulta de dados RDF.

5.4 Sistema Verificador de Ontologias baseado em Agentes

Esta seção mostra a proposta de Sistema Multiagente (MAS), apresentando também os seus componentes. Além disso, é apresentado como os agentes são criados e a integração entre

[1] Grupo de Otimização da Web (www.grow.ic.ufal.br)

ontologias e agentes. O principal objetivo desta seção é apresentar a arquitetura macro do MAS. Assim, esta arquitetura é composta por: i) um Agente Gerente *Manager Agent*, ii) uma Sociedade agente *Agent Society*, iii) uma máquina de inferência *Rule Engine*; iv) Regras *Rules* e v) as ontologias *Ontologies*, como pode ser observado na Figura 5.5. Cada um destes foi detalhado abaixo:

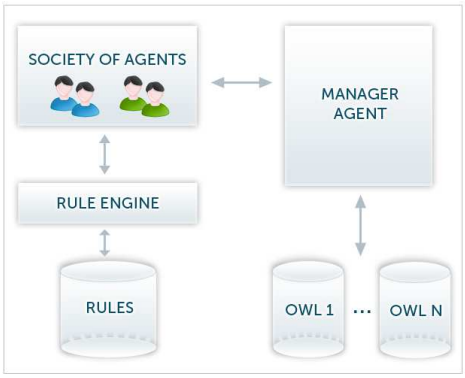


Figura 5.5: Multiagent Architecture

- **Manager Agent** - Agente Gerente é responsável por construir e inicializar a sociedade de agentes*Agent Society*. Este agente verifica o diretório em busca de novas ontologias. Para cada nova ontologia encontrada, este agente descobre que tipo de ontologia (isto é, o agente verifica se a ontologia é um produto ou uma ontologia LPS). Todas as ontologias apresentadas no diretório têm um agente que corresponde na sociedade.

- **Agent Society** - Esta sociedade é composta por agentes heterogêneos e que contém dois tipos de agentes: i) Agentes Produtos *Product Agents*, são responsáveis pela gestão das ontologias dos produtos, e ii) Agentes LPS *SPL Agents*, que são responsáveis pela gestão da ontologia de LPS.

- **Rule Engine** - Esta camada é responsável pela execução de regras na ontologia. Ele usa motor de inferência para ontologias de processamento oferecido pelos agentes na sociedade.

- **Rules and Ontologies** - esta camada é dividida em duas partes: i) esta parte contém as regras de verificação para validar o software restrições linha de produtos, e ii) que contém os arquivos de ontologia que será acessado pelo agente gestor.

Capítulo 6

Avaliação e Discussão

Este capítulo tem por objetivo apresentar uma avaliação da abordagem de linha de produto proposta, tendo em conta o investimento em dois domínios: Programação e Física, sendo que apenas o primeiro foi utilizado na descrição seguinte. Para tanto, descreve-se principalmente como os modelos são utilizados com o propósito de construir Sistemas Tutores Inteligentes em Larga Escala. Além disso, está descrito como foram modeladas as instâncias/conceitos nas ontologias de domínio e nas ontologias da linha de produto. Assim, na Seção 6.1 será apresentado como foi instanciado o tutor de programação. Na Seção 6.2 será apresentado como foi a implantação destes tutores, que aconteceu no Instituto Federal, campus Palmeira dos Índios. Na Seção 6.3 será apresentado como foi possível implementar a evolução da linha de produto. E por fim, na Seção 6.4 está apresentado a discussão da avaliação da linha de produto implementada.

6.1 Sistema Tutor Inteligente para o Domínio de Programação

Esta Seção descreve como foi construído o Sistema Tutor Inteligente de Programação (o qual será referenciado apenas como tutor no restante deste capítulo). Primeiramente, será apresentada a modelagem do domínio de programação. Em seguida, será apresentada o processo de instanciação do produto através da ferramenta de autoria. E por fim, será demonstrado como o produto foi implantando.

6.1.1 Descrição do Domínio de Programação

Nesta seção está descrito o tutor de programação foi elaborado neste estudo de caso. Dessa forma, fez-se necessário descrever os aspectos relacionados ao domínio e aos aspectos pedagógicos:

- Descrição do Domínio: Neste módulo estão as informações a respeito do que será passado para os alunos. No trabalho aqui apresentado o domínio corresponde a assuntos relacionados à programação em linguagem C. O tutor adota uma abordagem pedagógica de aprendizagem baseada em resolução de problemas, onde os problemas são expressões que podem conter operações, tais como: *Entrada de Dados*, *Saída de Dados*, *Estrutura de Programa em C*, *Comando de Atribuição*, conforme a Figuras 6.1 e 6.2.

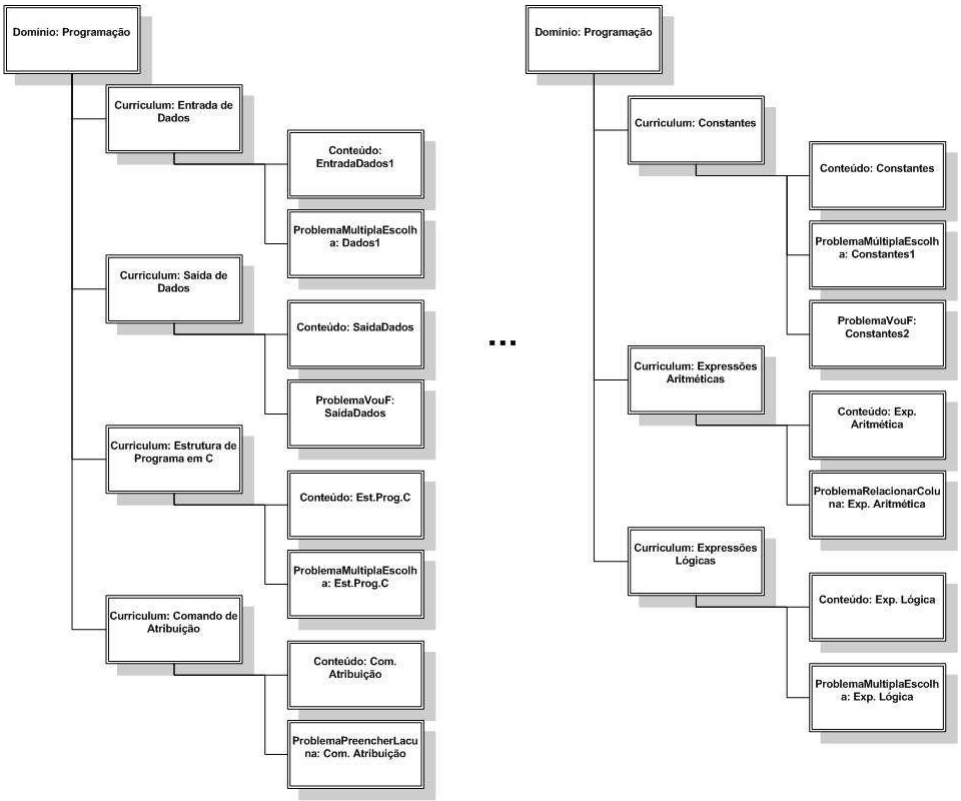


Figura 6.1: Descrição do Domínio

- Definição dos Conteúdos: Conforme as Figuras 6.1 e 6.2, os conteúdos foram especificados conforme os seus currículos. Então, por exemplo, para o currículo *Entrada de*

Dados, foi especificado um problema múltipla escolha (*Dados1*) e também um foi especificado um conteúdo (*EntradaDados1*). Os demais, conforme a Figura 6.2, também tem especificado um conteúdo e um problema.

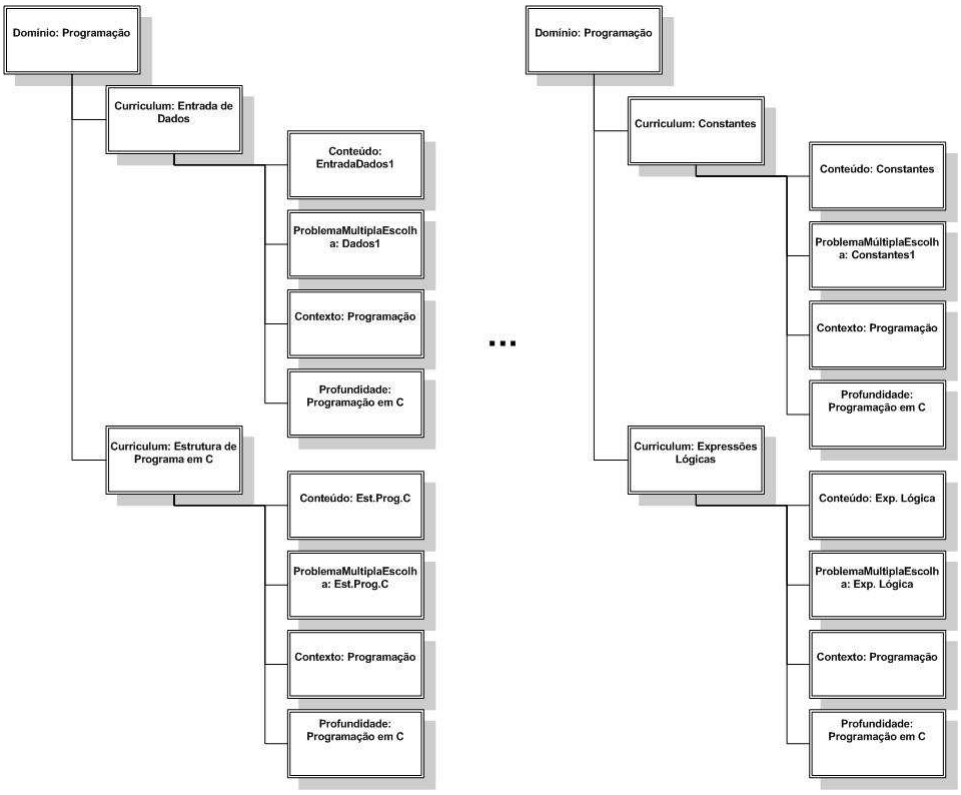


Figura 6.2: Descrição e Mapeamento dos Conteúdos

- Descrição Pedagógica: este módulo faz o gerenciamento do sistema tutor inteligente, tendo a função de coordenar o funcionamento do sistema. Além disso, o mesmo contém estratégias e táticas de ensino, selecionando o conteúdo a ser apresentado ao aluno, monitorando, criticando o desempenho do aluno, além do fornecer assistência quando solicitado. Quando uma questão for proposta para que o estudante a resolva, a interface recebe esta questão através do Módulo Pedagógico que por sua vez obtém do Módulo do Domínio. O Módulo Pedagógico comunica-se diretamente tanto com o Módulo do Domínio quanto com o Modelo do Aprendiz. Essa última comunicação ocorre, por exemplo, quando o usuário solicita a visualização do seu estado de aprendizagem. Desta forma, este módulo deve manter informações sobre a estratégia pedagógica adotada. Esta estratégia pedagógica adotada equivale a Aprendizagem Baseada em

Problemas[1][96]. A partir do momento que a estratégia pedagógica é definida, deve-se definir os pesos que fazem com que o estudante mude de um currículo para outro. conforme a Figura 6.3.

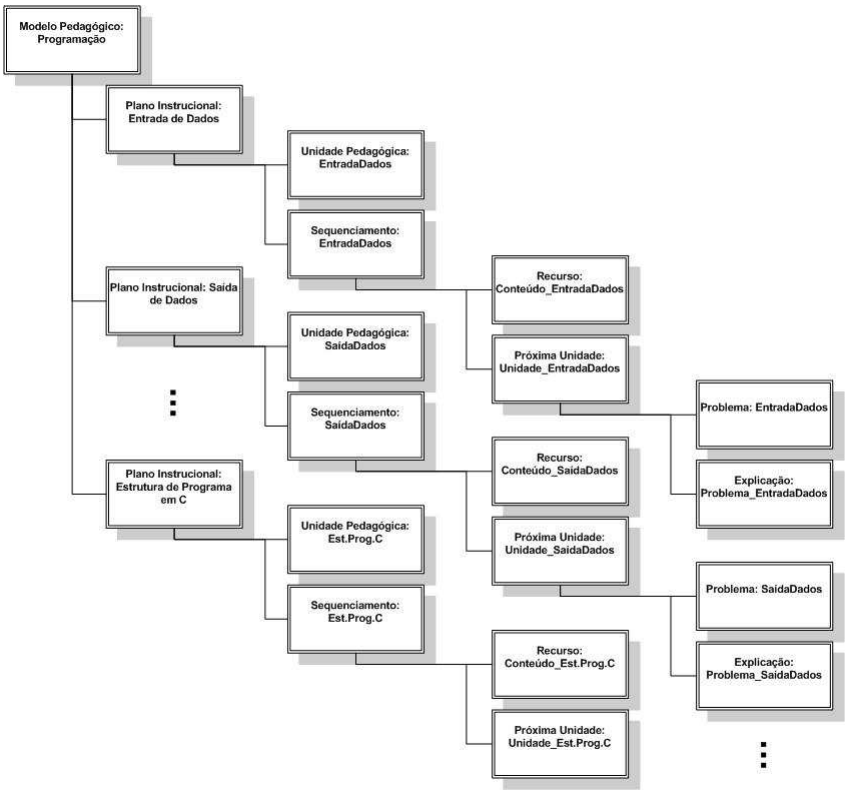


Figura 6.3: Descrição Pedagógica

6.1.2 Casos de Uso

Primeiramente foi definido o cenário em termos de casos de uso para o tutor de programação. O Cenário pré-estabelecido está descrito na Figura 6.4. Nesse cenário, pode-se identificar dois atores, um usuário, ou seja, um aluno e o próprio sistema. Assim, entre as principais atividades que um aluno pode realizar, pode destacar:

Fazer Login: O estudante é quem faz o seu próprio *login* no sistema. Isso é importante para que o sistema possa identificar quem é o estudante que está acessando o sistema em um determinado momento;

[1]Do Inglês, Problem-Based Learning

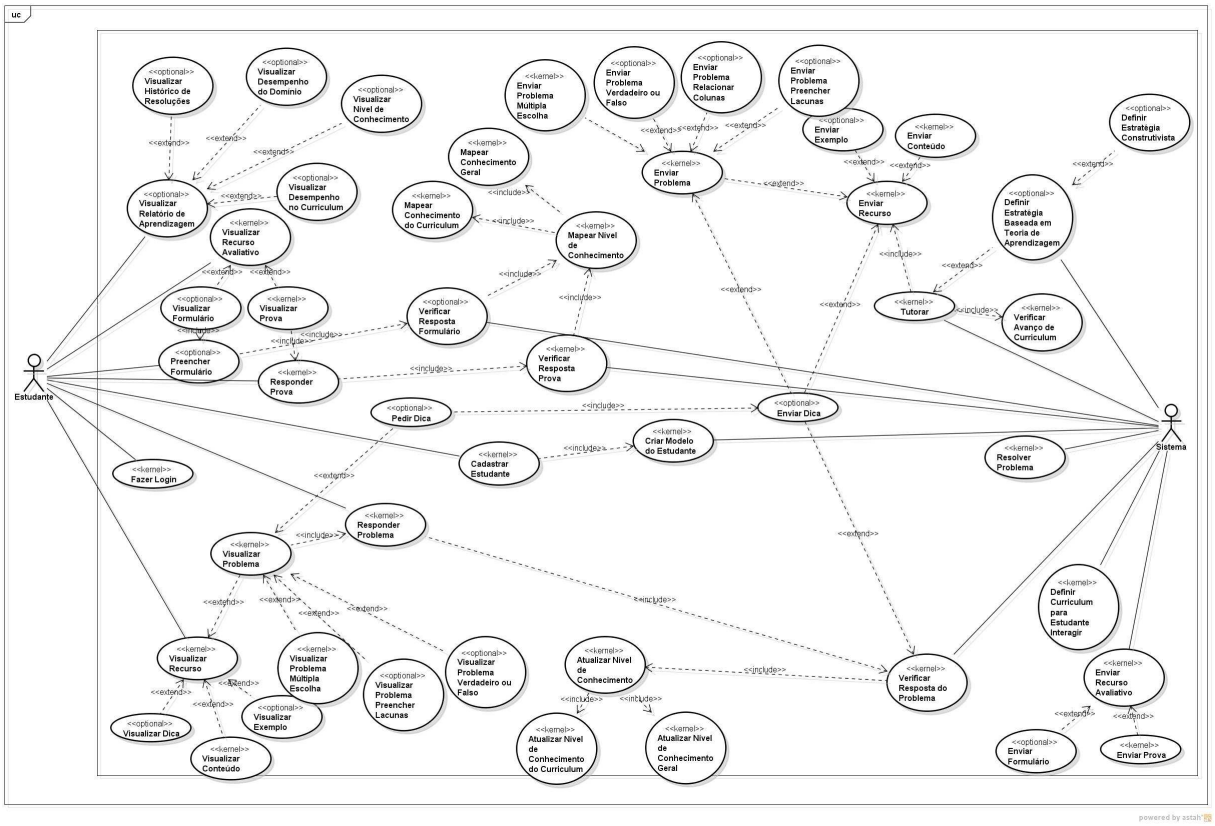


Figura 6.4: Diagrama de Casos de Uso Especializado para o Tutor de Programação

Visualizar Recurso: O estudando também está apto a visualizar recurso. Através disto, o estudante pode passar para um processo de ensino-aprendizagem. Isso ocorre porque são os recursos que proporcionam o ensino aos alunos;

Visualizar Recurso Avaliativo: Para que o sistema possa acompanhar de forma mais acurada o nível de conhecimento que o aluno se encontra, faz-se necessário que o mesmo, a cada período, se submeta a *Responder Prova*, de tal forma que o sistema poderá realizar suas tarefas com mais informação. Além disso, existe o *Preencher Formulário*, onde está ação é importante para que o sistema possa identificar com mais propriedade o aluno, fazendo-se necessário que o aluno forneça algumas informações iniciais, de tal forma que o sistema poderá mapear com mais precisão o nível de conhecimento que aluno se encontra;

Cadastrar Estudante: O processo de cadastro do estudante também é disponibilizado para cada aluno, de tal forma, que o próprio aluno é quem faz o seu cadastro no sistema,

colocando as suas informações pessoais de uma forma responsável;

Resolver Problema: O Aluno também pode responder problemas, isso é importante para que o aluno possa se exercitar, refletindo sobre os seus conhecimentos, para tentar resolver problemas.

O Sistema também possui as suas responsabilidades no processo de ensino aprendizagem. Pois o mesmo precisa acompanhar regularmente cada ação do estudante, e por isso, faz-se necessário que o mesmo também realize suas atividades, tais como:

Definir Curriculum para Estudante Interagir: O Sistema tem a função de definir o currículo para cada estudante. Isso é importante do ponto de vista de personalização, pois cada estudante pode ter níveis de conhecimento diferentes.

Definir Estratégia Beseada em Teoria de Aprendizagem: Cada estudante pode se adequar a estratégias pedagógicas diferentes de formas diferentes. No entanto, foi definido que o tutor só teria uma única estratégia pedagógica, onde a mesma seria utilizada indistintamente por todos os alunos;

Enviar Recurso Avaliativo: Esse tipo de abordagem é interessante, e o sistema pode recorrer ao mesmo quando for necessário identificar com uma melhor qualidade o nível do aluno;

Resolver Problema: Existem situações onde o estudante poderá não resolver um determinado problema, mesmo depois dos vários mecanismos de ajuda que o mesmo poderá ser submetido. Dessa forma, em último caso, o sistema poderá ajudar o estudante a resolver um determinado problema;

Tutorar: Este procedimento é o mais importante do sistema. Pois o mesmo é responsável por recomendar adequadamente os recursos educacionais aos estudantes. Este procedimento é realizado levando-se em consideração tanto o currículo do estudante, quanto a estratégia pedagógica que o mesmo foi submetido.

6.1.3 Diagrama de Features

Como colocado no Capítulo 2, mais precisamente na Seção 2.2.3, um diagrama de *feature* proporciona uma visão global da Linha de Produto no que diz respeito as variabilidades e comunalidades da mesma. Assim, torna-se possível perceber os pontos variáveis da arquitetura do sistema, ou melhor, da Linha de Produtos. Com isso, a partir do diagrama geral da Linha de Produto, conforme Figura 4.7, foi produzido uma linha de produto para tutores de programação. Abaixo segue a figura que exibe o diagrama de feature especifico desta linha.

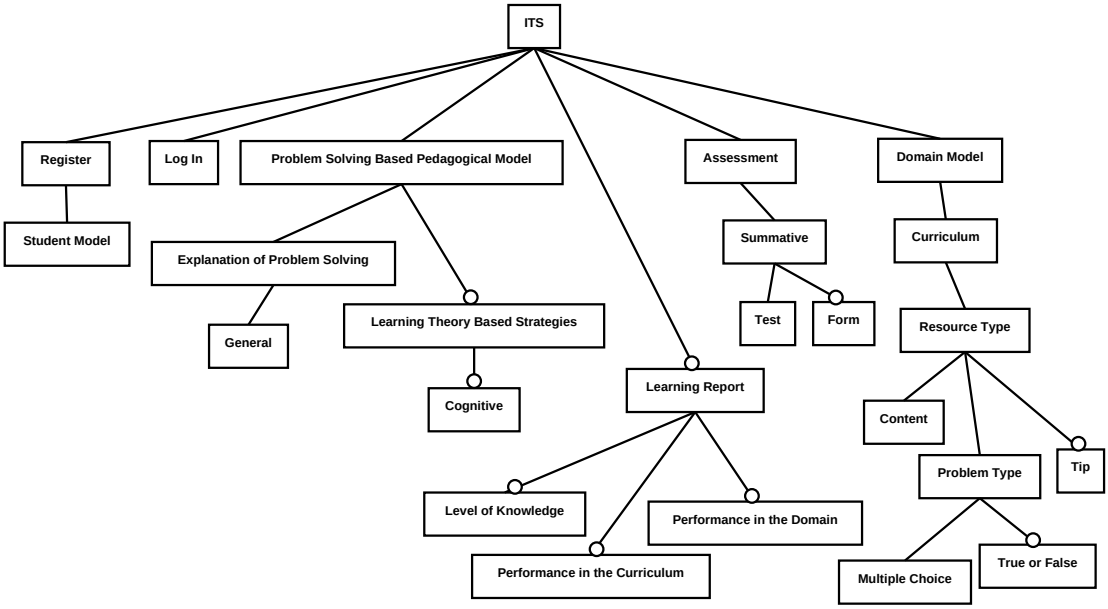


Figura 6.5: Diagrama de Features Especializado para o Tutor de Programação

Comparando os dois diagramas (Figura 4.7 e Figura 6.5) percebe-se que o STI desenvolvido tem todas as *features* obrigatórias como previsto. Além do mais, das *features* opcionais algumas foram providas, tais como:

- **Pedagogical Strategy - Cognitive:** Está *feature* está diretamente relacionada a uma questão essencial de um tutor: *Como ensinar?* Assim, está tem por função proceder dentro do tutor sequenciando recursos educacionais adequadamente para cada perfil de estudante: (i) Primeiramente apresentando conteúdo a cada estudante, (ii) em seguida, depois do estudante ler o conteúdo previamente enviado, o tutor submete um problema adequado para o mesmo;

- **Form**: Este recurso é importante para estabelecer um conhecimento inicial sobre cada

estudante;

- **True or False Problem**: Este é um tipo de problema que é importante para avaliar questões mais pontuais de um determinado currículo;
- **Hint**: Este recurso é importante para dar uma dica quando o estudante estiver passando por algum problema, ou para dar uma ajuda inicial quando o estudante não estiver sabendo como proceder para iniciar a resolução de um determinado problema;
- **Domain Learning Report**: Este relatório é importante para ajudar os professores a diagnosticar os problemas que estiverem ocorrendo com a turma.

É importante citar que em nível de implementação, cada *feature* foi mapeada em um componente de software. Através disto, pode-se adicionar nova *features* dinamicamente, sem um custo adicional de implementação em outros componentes.

6.1.4 Arquitetura

A arquitetura definida para o tutor de programação foi a que está expressa na Figura 6.6. Nesse sentido, a arquitetura resultante da escolha das features, da subseção anterior, refletem-se basicamente nos cinco pontos de variabilidade abaixo:

Relatórios: Existem alguns tipos de relatórios possíveis, e é possível escolher qualquer combinação deles para ser disponibilizado por um tutor aos seus usuários. Neste caso, foram escolhidos três: *Nível de Conhecimento*, *Performance no Currículo* e *Nível de Conhecimento*;

Tipos de Problema: Do ponto de vista dos tipos de problemas, também é possível especificar qualquer combinação dos tipos de problemas, No entanto, para o domínio do tutor, foram escolhidos os tipos *Múltiplas Escolhas* e *Falso ou Verdadeiro*;

Recursos Avaliativos: Os recursos avaliativos são meios que o *kernel* de um tutor tem para avaliar o nível de conhecimento de um determinado aluno. Então, o tutor tem o recurso *Teste* como obrigatório e o recurso *Formulário* como opcional;

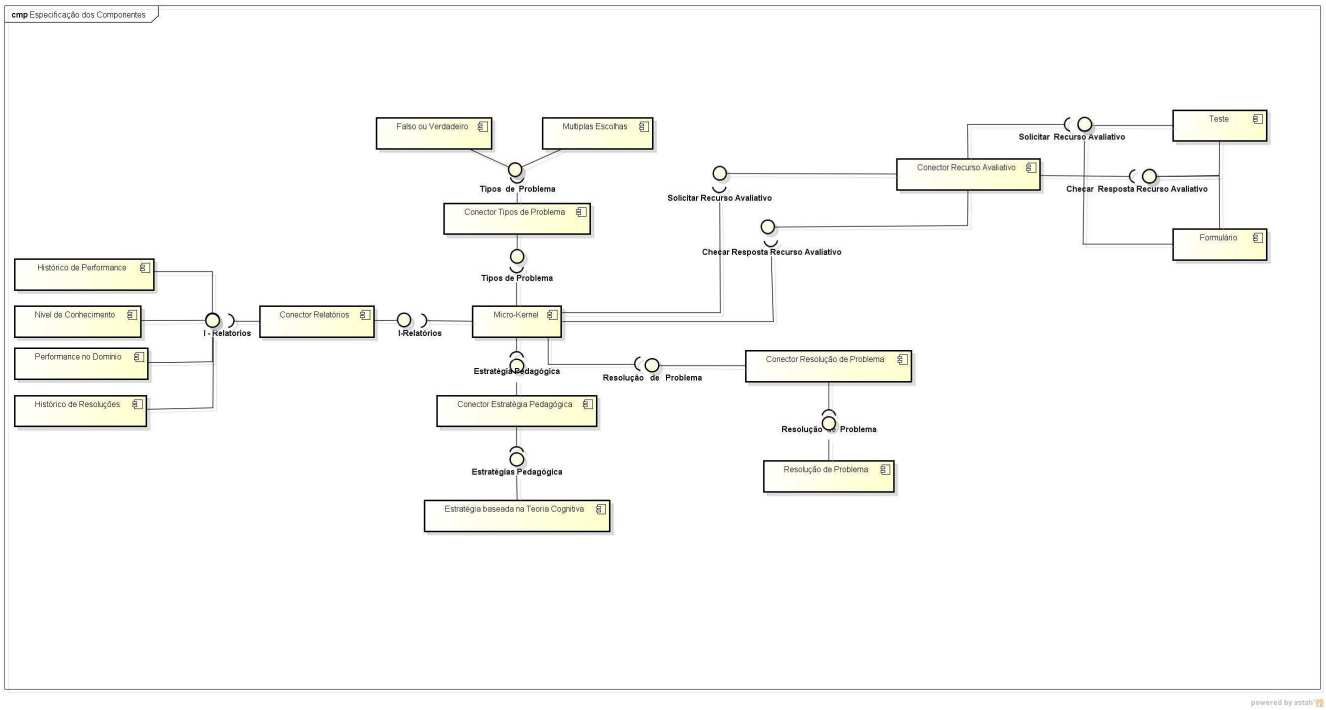


Figura 6.6: Diagrama de Componentes Especializado para o Tutor de Programação

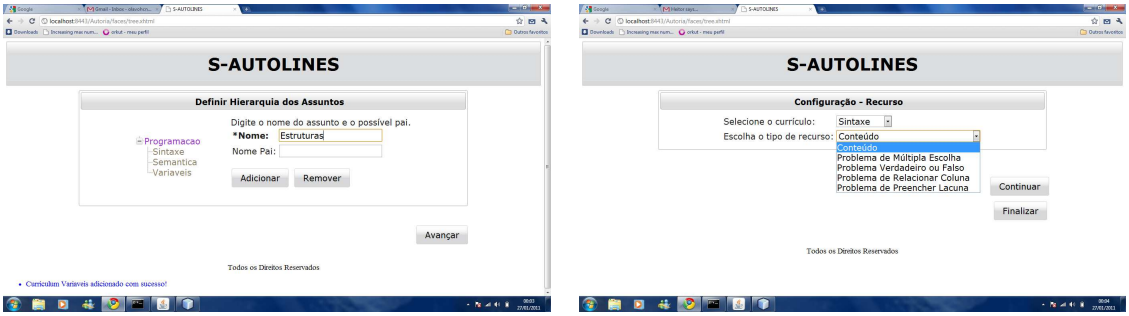
Resolvedor de Problemas: Os resolvedores de problemas são importantes e utilizados quando os alunos não conseguem, de uma forma definitiva, resolver um determinado problema que foi submetido a ele. Dessa forma, foi especificado o resolvedor geral de problemas;

Estratégias Pedagógicas: As estratégias pedagógicas são importantes no sentido de proporcionar o recurso educacional correto ao estudante. Nesse sentido, podem ser escolhidas quaisquer combinações, ficando por conta do *kernel* do tutor definir qual será utilizada por cada estudante.

6.1.5 Autoria do Domínio

Uma linha de produto pode viabilizar sistemas tutores inteligentes independente de domínio. Para isto, fez-se necessário construir uma ferramenta que dê suporte a especificar os domínios de conhecimento, de tal forma que a linha de produto e os seus produtos possam interpretar, do ponto de vista computacional, o que foi descrito. Assim, são necessário quatro passos para especificar o domínio:

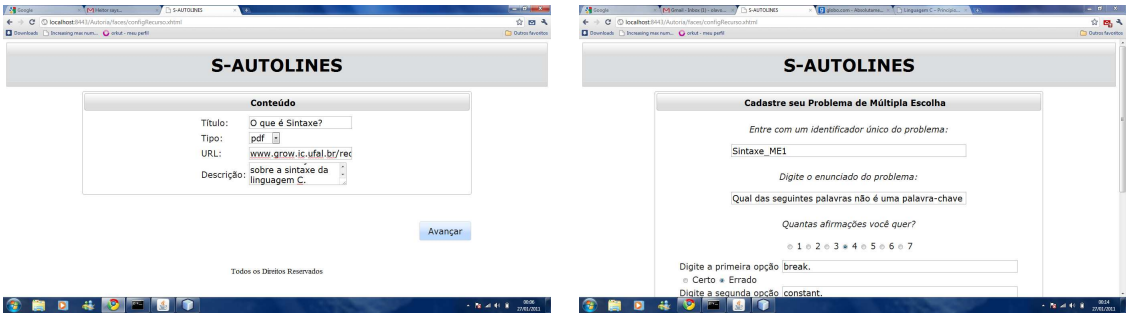
Especificação da Hierarquia do Domínio: Essa etapa consiste na definição dos currículos que um determinado domínio deve ensinar. Além disso, faz-se necessário o conhecimento do domínio, de tal forma que este possa ser dividido em partes adequadas do ponto de vista didático. Esta etapa está descrita na Figura 6.7(a);



(a) Definição dos Currículos (b) Cadastro dos Recursos

Figura 6.7: Ferramenta de Autoria do Domínio I

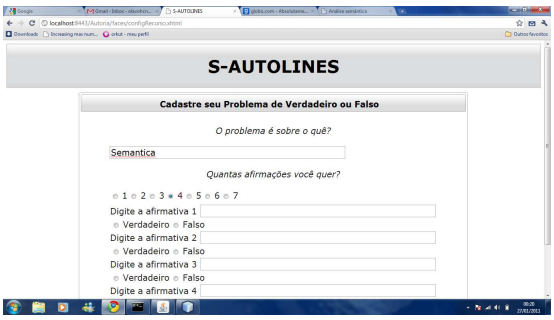
Cadastro de Conteúdo: Os conteúdos são importantes para que o aluno possa estudar o domínio que o mesmo irá se submeter a aprender. Assim, faz-se necessário ter conteúdos para os currículos previamente cadastrados. Esta etapa está demonstrada nas Figuras 6.7(b) e 6.8(a);



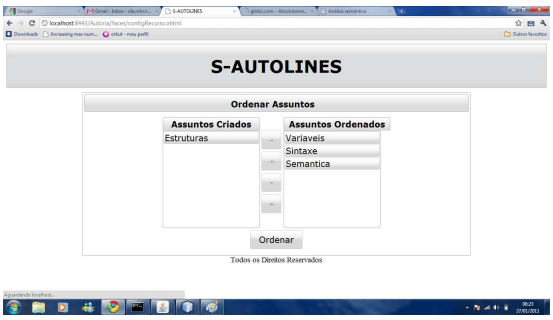
(a) Cadastro dos Conteúdos (b) Cadastro de Problemas Múltipla Escolha

Figura 6.8: Ferramenta de Autoria do Domínio II

Cadastro de Problemas: Os problemas são cadastrados como recursos dentro da linha de produto. Os problemas podem vir antes ou depois dos conteúdos relacionados, quem define são as estratégias pedagógicas. Assim esta etapa está demonstrada nas Figuras 6.7(b), 6.8(b) e 6.9(a);



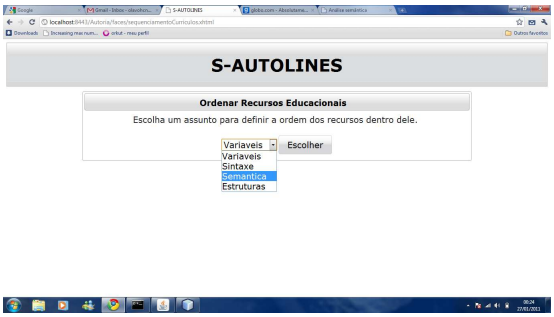
(a) Cadastro de Problemas do tipo Falso ou Verdadeiro



(b) Ordenar Currículos

Figura 6.9: Ferramenta de Autoria do Domínio III

Ordenar Currículos: Os currículos devem ser ordenados para que os sistemas tutores possam ter uma ordem padrão, que pode ser adotado por alguns sistemas tutores. Assim esta etapa e apresentada na Figura 6.9(b);



(a) Ordenar Recursos Educacionais

(b) Exportar Domínio

Figura 6.10: Ferramenta de Autoria do Domínio IV

Ordenar Recursos Educacionais: Depois de ordenar os currículos, deve-se ordenar os recursos educacionais. Isso é importante para que o tutor possa conhecer a ordenação padrão fornecida pelo autor do domínio.Assim esta etapa está demonstrada nas Figura 6.10(a);

Exportar Domínio: Por fim, quando o domínio for definido, o autor pode exportar o mesmo, para que ele seja embutido em alguma linha de produto. Assim esta etapa está demonstrada na Figura 6.10(b);

6.1.6 Ferramenta de Autoria do Produto

Um dos principais objetivos de uma ferramenta de autoria de STIs é tornar o processo de construção dos tutores inteligentes simples, permitindo que um número maior de projetistas, de diversas áreas, possam utilizá-la. Nesse sentido, 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) que reduza a quantidade de conhecimento necessário para utilizá-la. Podemos citar como um elemento da GUI que facilita a realização de uma tarefa o wizard. Este elemento da GUI, 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. A GUI da ferramenta de autoria, mostrada abaixo, baseia-se em um "wizard" onde o processo da autoria é composto por passos simples até chegar ao ponto de implantação do sistema.

O processo de autoria de um STI é dado por seis passo: *User Login*, *SPL Selection*, *Product Customization*, *Feature Validation*, *Product Generation* e *Product Deployment* melhor descritos a baixo:

1. **User Login:** antes de iniciar o processo de autoria efetivamente é necessário que o usuário, que já esteja cadastrado, seja autenticado pelo sistema. Para tanto o usuário deve informa seu login e sua senha e clicar no botão "login" (como mostrado na Figura 6.11) ou se ainda não estiver cadastrado clicar no botão "*New Registration*" e repetir o processo descrito anteriormente.

Figura 6.11: tela de login do usuário (User Login)

2. **SPL Selection:** neste passo o projetista deve selecionar a linha de produtos de software (LPS) e o nome do produto (tutor) a ser ser gerado. Para tanto, deve-se informar o nome do tutor através do campo "*Product Name*", escolher a LPS entre as opções existentes em "*Software Product Line (SPL)*" e clicar no botão "*Customize Product*" (vide Figura 6.12).

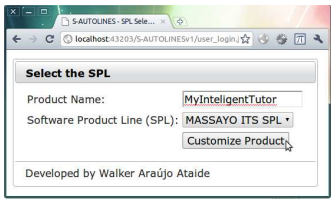


Figura 6.12: Tela de seleção da SPL a ser utilizada.

3. **Product Customization:** neste passo o projetista deve customizar o tutor inteligente a partir do *feature model* da linha de produto de software escolhida no passo anterior (*SPL Selection*) (Figura 6.13). Para tanto, deve-se clicar no *checkbox* para selecionar a *feature* que deseja incluir no seu tutor. As *features* podem ser "*mandatory*" que devem ser obrigatoriamente marcadas, "optional"que são opcionais e "alternative"que caso seja marcada não pode marcar outra feature alternativa a ela. Os tipos das features são descritos entre parêntese no seu lado direito.

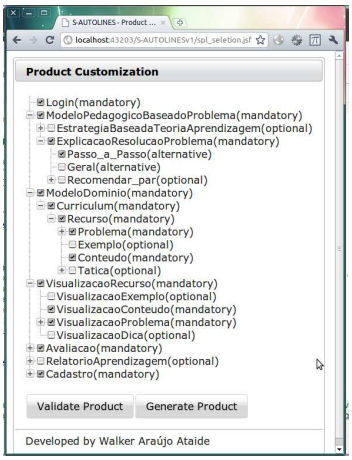


Figura 6.13: Tela de customização do tutor

4. **Feature Validation:** neste passo o projetista deve clicar no botão "*Validate Product*" para que a ferramenta de autoria verifique se há erros na seleção das *features*, caso existam erros, a ferramenta informa ao projetista para que sejam corrigidos (vide Figura 6.14), caso contrário, a ferramenta emite uma mensagem de que o produto está correto (vide Figura 6.15) e permite que o projetista possa gerar o produto (*Product Generation*).

5. **Product Generation:** para realizar este passo o projetista deve ter validado as *features*

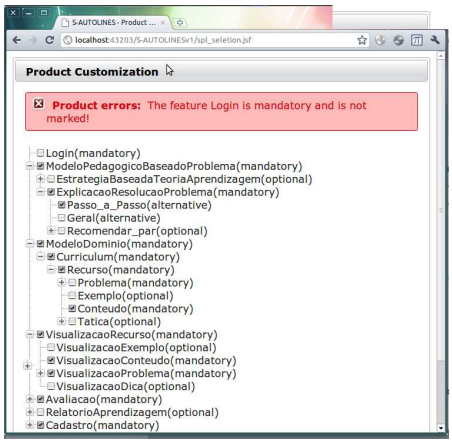


Figura 6.14: Tela de customização do tutor exibindo os erros encontrados no feature model

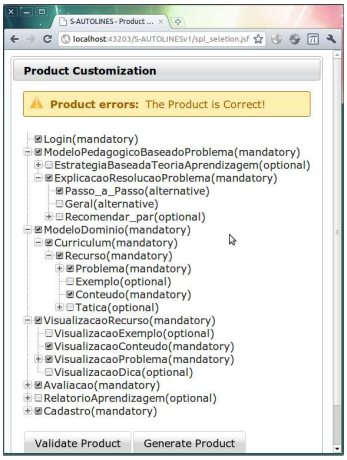


Figura 6.15: Tela de customização do tutor exibindo a mensagem de que o produto está correto

(como mostrado no passo anterior) recebido a mensagem que o produto está correto (vide Figura 5), então deve clicar no botão "*Generate Product*" (vide Figura 5) para que a ferramenta de autoria gere o STI configurado, no caso de um STI que utiliza tecnologia Java web a ferramenta irá gerar o arquivo WAR com os módulos das features selecionadas.

6. **Product Deployment:** este passo, na realidade, é realizado pela própria ferramenta que implanta o STI no servidor (vide Figura 6.16), porém o projetista deve copiar a url informada na mensagem (vide Figura 6.17) e colá-la no navegador de internet para que possa acessar o tutor gerado e implantado (vide Figura 6.21) concluindo assim o processo de autoria do STI.

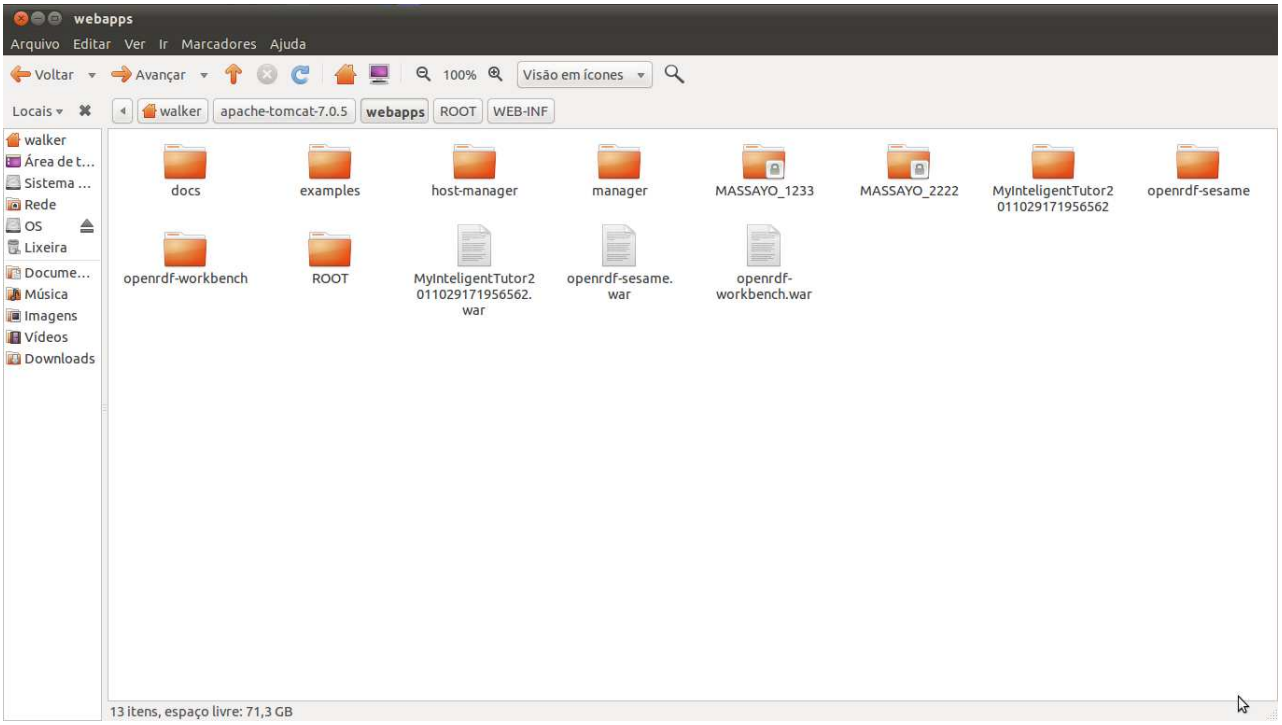


Figura 6.16: Pasta webapps do servidor apache tomcat 7.0.5 com tutor implantado

Os passos utilizados para instanciar um STI através da ferramenta de autoria estão intimamente ligados à criação, persistência e alteração da ontologia que o descreve em termo de suas *features*. Abaixo são descritas as etapas da instanciação da ontologia do STI e estão relacionadas aos passos do processo de autoria:

1. **Criação da Ontologia:** a ontologia do STI é criada no passo "SPL Selection". É gerado um arquivo OWL (vide Figura 6.19) que descreve a ontologia que representa o tutor a ser gerado.

2. **Persistência da Ontologia:** após a criação da ontologia ela é persistida em um repositório de ontologias como o SESAME (vide Figura 6.20) .

3. **Alteração da Ontologia:** no passo "*Product Customization*" quando o projetista marca (o estado da *feature* é alterado para "*SELECTED*") ou desmarca (o estado da feature é alterado para "*ELIMINATED*") os *checkbox* do feature model, a ontologia do STI reflete essas alterações tanto no arquivo OWL da ontologia (utilizado para verificação) quanto no repositório de ontologias (vide Figura 6.21).

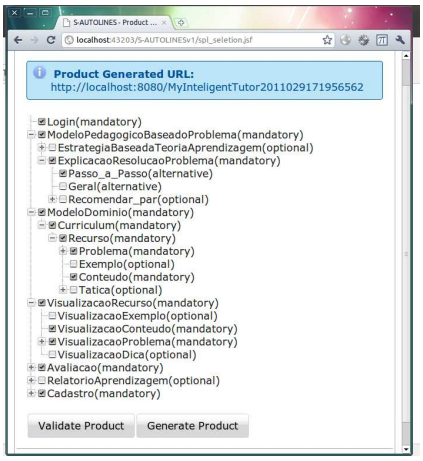


Figura 6.17: tela de customização do tutor exibindo a mensagem com a url do tutor gerado

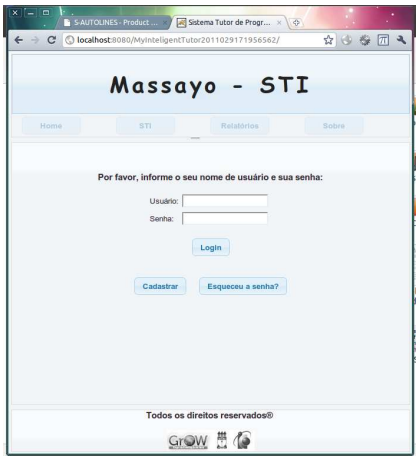


Figura 6.18: Navegador web mostrando o STI implantado e funcionado

6.1.7 Instâncias da Ontologias de Linhas de Produto

De acordo com as ontologias descritas no Capítulo 4, é possível criar diversos produtos derivados destas. Desse modo, foi criada uma ontologia de produto que especifica exatamente o diagrama de *feature* de tutor já apresentado. Nesse sentido, como primeira demonstração, serão apresentadas aqui algumas instâncias da ontologia do produto para melhor entendimento da configuração da linha de produto.

Assim sendo, a Figura 6.22 ilustra uma feature *LearningTheoryBasedStrategies* que é pai das seguintes features: *Cognitive*, *Constructivist*, *Socratic* e *Behaviorist*. A *LearningTheoryBasedStrategies* representa as estratégias pedagógicas que podem ser utilizadas em um determinado produto, que no nosso caso é um tutor. Como pode ser visto nesta figura, a

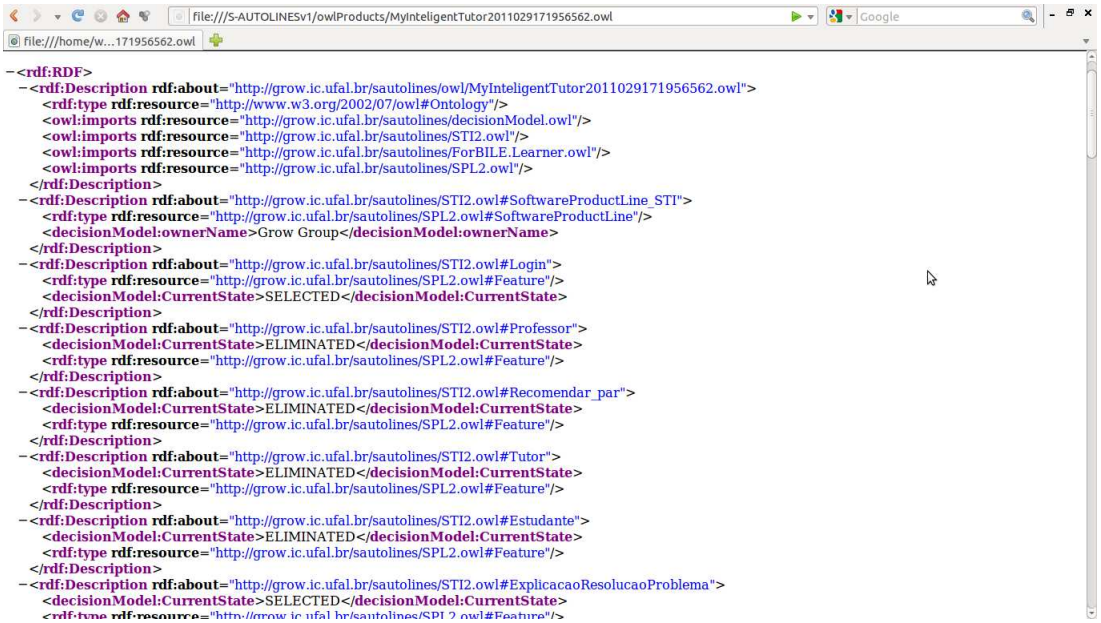


Figura 6.19: Arquivo OWL gerado pela ferramenta de autoria

única *feature* filha que foi selecionada para o tutor é *Cognitive*, por isso, para esta *feature*, o sistema segue os passos da estratégia pedagógica cognitiva comentado anteriormente.

Segundo a configuração da ontologia, será analisado outras *features* que são também importantes para a construção de um tutor decorrente da modelagem e, conseqüentemente, de sucesso. Essas features fazem parte da resposta da seguinte pergunta: Como ensinar?, elas representam os tipos de problemas que serão expostos para os estudantes resolverem. Para cada usuário/estudante existe um tipo de problema mais adequado tanto para fazê-lo aprender quanto desafiar a fazê-lo, isto é, estimulá-lo. A Figura 6.23 mostra como isto está modelado na ontologia do tutor.

Pode-se notar que para qualquer produto da linha de produto de software para ITS precisa ter pelo menos um tipo de problema que é *MultipleChoiceProblem*, por isso que na figura a *feature* está anotada como *mandatory*. Analisando-se um pouco mais, percebe-se também que o tutor de programação possui dois tipos de problemas (Múltipla Escolha e Verdadeiro ou Falso).

Detalhando o processo de configuração do produto, pode-se notar: (1) Construir uma linha de produto (específica) baseado na ontologia SPL, (2) Definir as restrições e obrigatoriedades para cada *feature* utilizando a ontologia *decisionModel* e (3) por último, construir os produtos baseado nas definições anteriores.

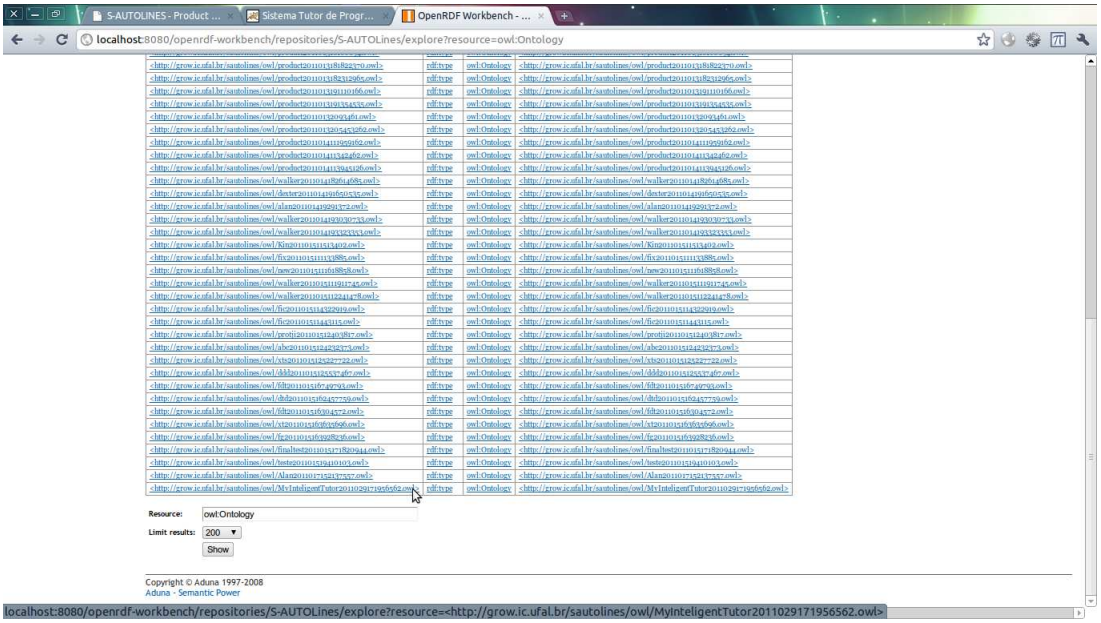


Figura 6.20: Ontologia do STI persistida no SESAME

Assim, de acordo com a configuração, todas as *features* que tinham a propriedade *CurrentState* definida como *SELECTED* foram transformadas para componentes de software.

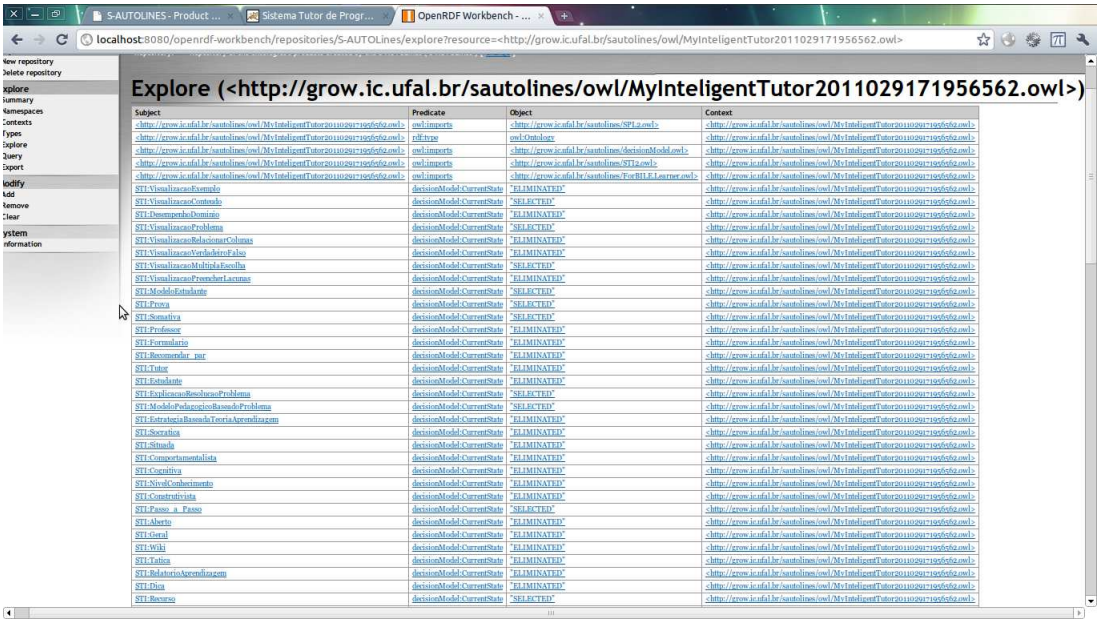


Figura 6.21: Ontologia do STI no SESAME com as alterações do estado das features

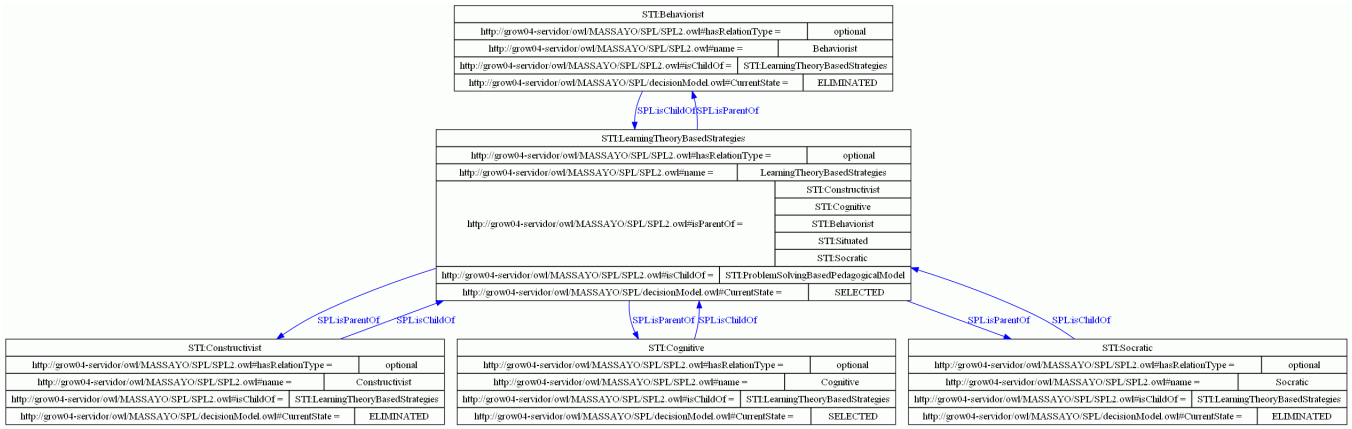


Figura 6.22: Estratégias Pedagógicas

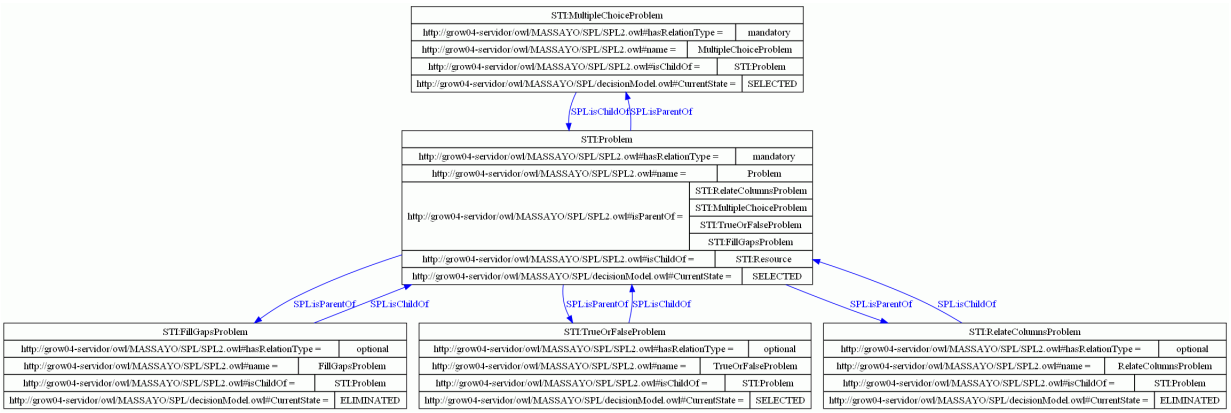


Figura 6.23: Tipos de Problema

6.2 Implantação

Esta seção tem por objetivo mostrar como foi implantado o sistema tutor inteligente de programação no Instituto Federal de Alagoas (IFAL) junto aos alunos de ensino técnico da mesma instituição. O mesmo está localizado no interior do estado de Alagoas no agreste no município de Palmeira dos Índios.

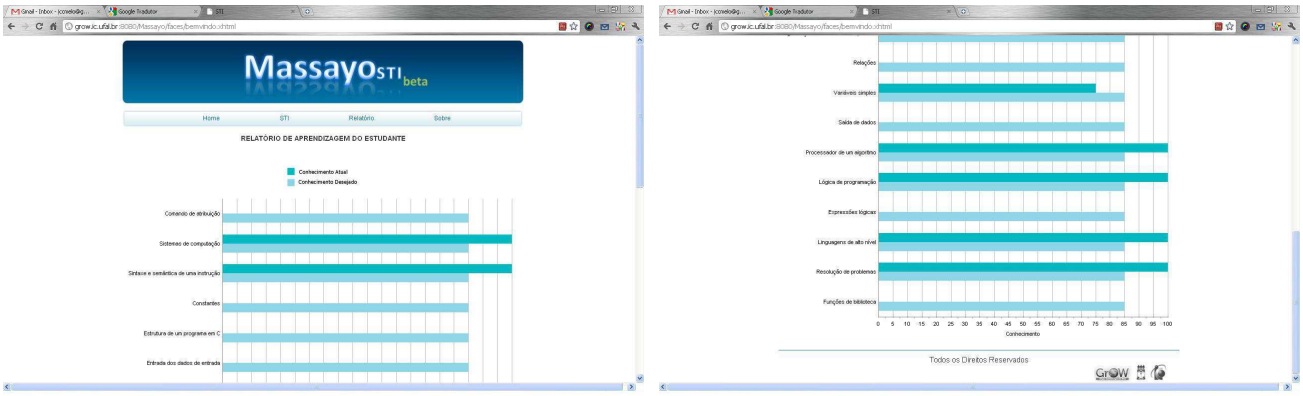
Este tutor foi desenvolvido no laboratório do Grupo de Otimização da Web (GrOW) num período de, aproximadamente, 15 dias. Neste período foi realizado todo o processo de modelagem do tutor descrito anteriormente. Mais adiante, na subseção seguinte, analisaremos como foi este processo de implantação por parte dos estudantes do IFAL e discutiremos a respeito disso.

A Figura 6.24 mostra a tela inicial do tutor de programação. A aplicação foi instalada no servidor da Universidade Federal de Alagoas, mais precisamente, no Instituto de Computação da Universidade para disponibilizá-la aos alunos. Assim, o tutor foi testado em um laboratório do IFAL por 10 alunos num período de 2 horas.

Figura 6.24: Tela Inicial

Quando o usuário se conecta ao sistema ele vai automaticamente para uma tela de bem-vindo, onde a mesma contém duas opções: (1) Estudar e (2) Ver Relatório de Aprendizagem. A primeira opção leva o usuário a interagir com o tutor, baseado em uma determinada es-

tratégia pedagógica, de forma a apresentar o conteúdo, depois o problema e assim, computar o nível de conhecimento do usuário a cada iteração. No entanto, a segunda opção transporta o usuário para uma página que contém todos os currículos de um certo curso/domínio, a porcentagem requerida para passar de curriculum e a porcentagem que o usuário está em cada curriculum no momento. As figuras 6.25(a) e 6.31(c) ilustram a segunda opção.



(a) 1 (b) 2

Figura 6.25: Tela dos Relatórios de Aprendizagem

Suponha que o usuário quis a opção (1) Estudar. Em seguida, o tutor sabe se ele se conectou pela primeira vez ou não, através de uma pesquisa na ontologia. Se ele está interagindo pela primeira vez, então o tutor exibe um formulário na tela para tentar descobrir o nível de conhecimento do usuário. Depois disso, o sistema mostra um conteúdo (dependendo da estratégia pedagógica) para o usuário e, depois, um problema. A Figura 6.26 apresenta um problema do tipo múltipla escolha para o usuário resolver. Além do mais, enquanto o usuário está no sistema estudando todas as telas, aparece uma barra de evolução posicionada imediatamente abaixo do cabeçalho da página que representa o nível de conhecimento do usuário num certo curso/domínio.

Uma equipe do GrOW foi para o IFAL a fim de oferecer um suporte aos estudantes como também obter suas respectivas considerações quanto ao tutor em questão. Além disso, a equipe levou consigo um formulário com vários questionamentos a respeito de diferentes conceitos, tais como: usabilidade, design/layout das paginas, autonomia no ensino, entre outros. A Figura 6.27 mostra um dos alunos testando o tutor.

Uma outra Figura 6.28 exibe dois membros do GrOW acompanhando as atividades realizadas por um dos alunos que estava interagindo com o tutor.

Figura 6.26: Tela de Problemas

Figura 6.27: Uma estudante que avaliou o tutor.

Após a implantação do tutor os alunos foram submetidos ao questionário, onde eles puderam avaliar como foi o desempenho do tutor. O resultado desta avaliação encontra-se na subseção 6.4.2.

Figura 6.28: Membros do GrOW acompanhando a avaliação.

6.3 Evolução

Esta seção mostra como foi realizada a evolução automatizada do STI de programação dentro dos três cenários colocados na Seção 4.7. A estrutura foi elaborada utilizando ontologias e agentes inteligentes. As ontologias foram importantes para que o conhecimento sobre a linha de produto e os próprios produtos pudessem ser interpretados pelos agentes de software, de tal forma que os mesmos possam ser atualizados conforme eventuais tipos de evolução, descritos nas subseções a seguir.

6.3.1 Cenário II: Button-up

Este segundo cenário acontece quando uma nova *feature* é desenvolvida para um determinado produto, e por isso, esta nova *feature* deve ser disponibilizada para a Linha de Produto. Com isso, no processo de criação de um novo produto, será possível utilizar esta nova *feature*. Isso é importante para que uma *feature* criada para um propósito, a priori, particular, possa ser distribuída para novos produtos. Por exemplo, um novo tipo de estratégia pedagógica é implementada para que seja fornecido sempre o conteúdo antes do problema.

Para realizar esta tarefa, foi implementado o *Agente Verificador de Atualização de Produto*, este agente tem por função atualizar a linha de produto com o novo componente.

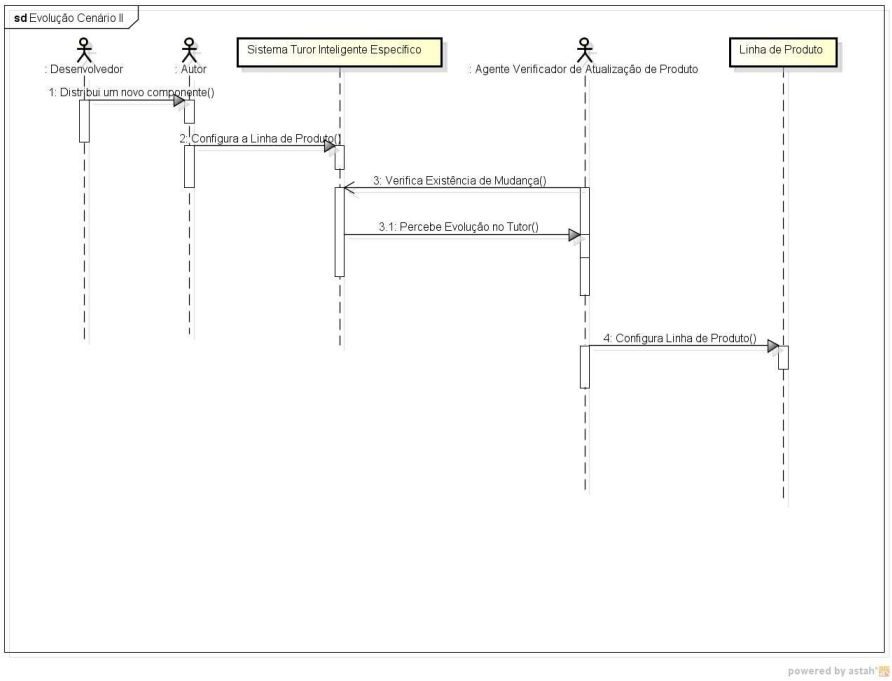


Figura 6.29: Diagrama de Sequencia do Cenário II

Assim, conforme o diagrama de sequencia da Figura 6.29, faz-se necessário os seguintes passos para que cada evolução seja realizada:

Passo 1: Primeiramente, um novo componente deve ser implementado, ou um componente adquirido por terceiros deve ser adaptado, para fornecer uma nova *feature*. Em seguida, este componente deve ser adquirido pelo autor de algum tutor, para que o mesmo possa ser embutido dentro do tutor;

Passo 2: Em seguida, o autor do tutor deve atualizar o seu tutor com a nova *feature*, por sua vez, o novo componente. Esse passo não requer conhecimento de como o tutor foi implementado;

Passo 3: O *Agente Verificador de Atualização de Produto* tem como objetivo observar mudanças, no sentido de evolução, em cada produto. Isso significa que o mesmo deve identificar quando uma nova feature foi adicionada;

Passo 4: Por fim, o *Agente Verificador de Atualização de Produto* atualiza a linha de produto, de tal forma que a nova *feature* possa ser utilizada em outros diferentes produtos.

6.3.2 Cenário III: Evolução baseado em Serviços Semânticos

Por fim, para verificar a evolução baseada em serviços semânticos, foi utilizado, primeiramente, o conector que define as estratégias pedagógicas. Este conector pode ser visualizado na interface *Pedagogical Strategies*. Esta interface especifica que o conector deve receber como entrada uma URI que especifica o estudante e uma URI que especifica o último recurso educacional enviado àquele estudante. Assim, com estas duas entradas, o conector repassa a chamada para uma dos componentes que se comporta como estratégia pedagógica. A primeira ontologia está definida na Figura 6.30 e 6.31, e a segunda ontologia está definida na Figura 6.32.

Para testar esta evolução automatizada, foi estabelecido o seguinte cenário: Primeiramente, foi criado um produto que utiliza uma estratégia pedagógica chamada *Problem Based Learning*. Este componente fornece instrução baseada em problemas, e o tipo de recurso educacional é alterado quando o estudante não consegue acertar uma determinada tarefa. No entanto, serviços semânticos podem ter sido disponibilizados na Internet, de tal forma que os mesmos podem ser utilizados para fornecer serviços interessantes dentro dos pontos de variabilidade do nosso tutor. Assim, foram criados dois serviços, que de forma combinada, podem ser utilizados para suprir uma necessidade.

Assim, o produto testado para construção do sistema tutor utiliza inicialmente o componente *PBL*, conforme seta C (Figura **??**), como o componente que tem a responsabilidade de processar qual o próximo recurso educacional que será enviado a um determinado estudante. No entanto, conforme a seta A, da Figura **??**, foi implementado um sistema multiagente que tem por função:

(i) Identificar mudança no uso de componente, pelos conectores. Ou seja, um conector pode se utilizar de um grupo de componentes para fornecer um determinado serviço;

(ii) Buscar na Internet por combinações de serviços que possam ser utilizados por algum, ou alguns, produtos;

(iii) Configurar o componente o meta-componente para poder utilizar, de forma combinada, os dois serviços que forma descobertos;

(iv) Atualizar a linha de produto para que os conectores semânticos possam utilizá-lo.

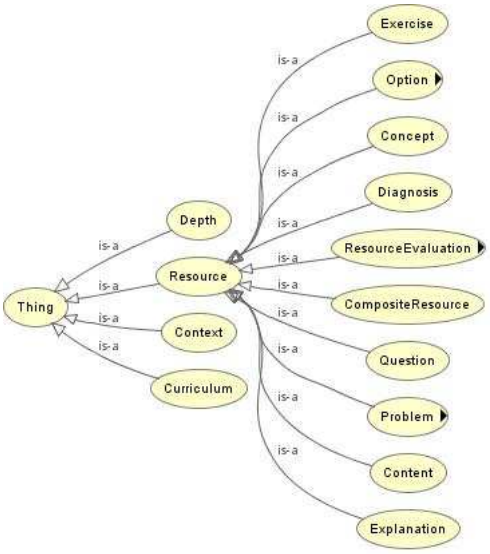
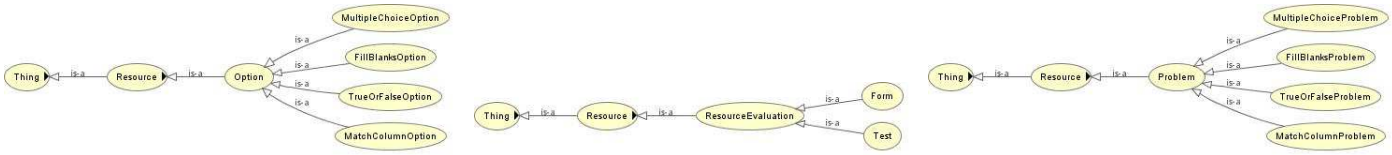


Figura 6.30: Ontologia que descreve Recursos Educacionais



(a) Classe Opção (*option*) (b) Classe Recurso de Avaliação (*ResourceEvaluation*) (c) Classe Problema (*Problem*)

Figura 6.31: Sub-tipos de Recursos Educacionais

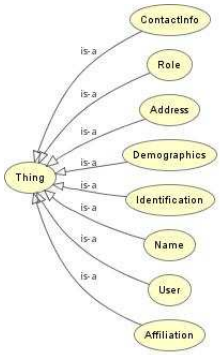


Figura 6.32: Ontologia que Descreve um Aprendiz

Uma vez que o meta-componente foi configurado para utilizar um grupo de serviços com o objetivo de fornecer recursos educacionais para um determinado aprendiz que esteja se submetendo a um processo de aprendizagem fornecido por um tutor, cabe ao meta-componente disponibilizar recursos educacionais conforme a demanda repassada por cada tutor. Além

disso, enquanto um componente de estratégia pedagógica tem que se utilizar de recursos educacionais pré-estabelecidos no repositório particular do tutor, o meta-componente de estratégica pedagógica pode fazer uso de recursos oriundos de serviços semânticos. No entanto, conforme estabelecido na interface da estratégia pedagógica, o retorno especificado pela interface é apenas a URI do recurso educacional, e isso implica que o tutor deve buscar este recurso no seu próprio repositório.

Por outro lado, se o recurso educacional sugerido pelo meta-componente não estiver localizado no repositório, o mesmo terá que: (i) Buscar o recurso educacional; e (ii) depositá-lo no repositório do tutor, para que o mesmo possa utilizá-lo. Dessa forma, para o tutor, a questão do recurso educacional ser oriundo de fontes externas torna-se transparente para o tutor. Sendo, portanto, responsabilidade do componente semântico realizar esta tarefa.

Na Figura 6.34 encontram-se os serviços semânticos implementados. Estes podem surgir de uma evolução do conhecimento, como uma evolução na descrição de recursos educacionais (Ontologias das Figuras 6.31 e 6.30) para uma descrição mais sofisticada, que permite inferências com mais precisão. Portanto, como ilustração, foi desenvolvida uma ontologia que descreve uma evolução, sendo a ontologia de Recursos Educacionais Especializados (conforme Figura 6.33). Assim, com a existência desta evolução, novos serviços semânticos podem ser desenvolvidos para proporcionar uma melhor evolução dos tutores, em consonância com o conhecimento evoluído.

Além disso, esses serviços tanto podem ser desenvolvidos pelos desenvolvedores dos tutores, quanto podem ser desenvolvidos por terceiros, por algum motivo específico. Foram desenvolvidos dois serviços (Serviço A e B, conforme Figura 6.34) que poderiam tratar a evolução da descrição dos recursos educacionais, conforme Figura 6.33. Como pode ser visto, o primeiro serviço (serviço A) recebe como entrada a URI de um aluno e a URI de um estudante, e retorna a descrição (URI) de um novo tipo de recurso, o recurso especializado. Com esta nova URI, é possível recuperar fisicamente um novo recurso, e também é possível recuperar a URI padrão deste recurso. Para isto, o componente semântico processa uma requisição em três etapas:

Etapa 1: Recebe uma requisição do sistema tutor;

Etapa 2: Repassa a requisição para o serviço semântico A, em busca da especificação no

Figura 6.33: Ontologia que Descreve um Recurso Educacional Especializado

novo recurso educacional a ser diponibilizado ao aprendiz;

Etapa 3: Com a resposta do serviço semântico A, da etapa anterior, o componente invoca o serviço semântico B, e coloca o novo recurso no repositório, e devolve para o tutor a URI do novo recurso.

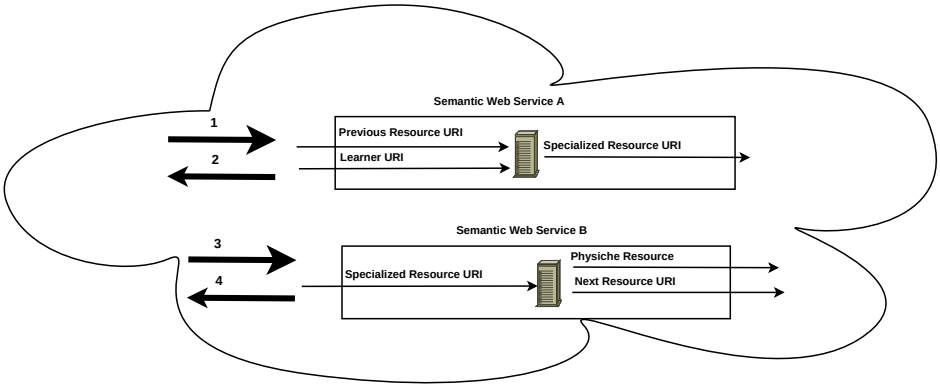


Figura 6.34: Serviços Semânticos Implementados

Do ponto de vista mais macro, é o *Agente Compositor de Serviços* que atualiza o produto e a linha, de tal forma que o nova composição dos serviços possa ser utilizada mais amplamente em novos produtos. Então, conforme está expresso no diagrama de sequencia 6.35, as interações ocorrem da seguinte forma:

Passo 1: O *Agente Compositor de Serviços* descobre uma composição de serviços, e depois disso, faz-se necessário configurar o meta-componente, para que o mesmo saiba quais serviços semânticos devem ser acessados, e em qual sequencia;

Passo 2: Em seguida, o *Agente Compositor de Serviços* precisa configurar o conector semântico para que o mesmo possa acessar o meta-componente, de forma adequada. Isso é importante porque o meta-componente pode possuir várias composições de

serviços, e por conta disso, faz-se necessário invocar o meta-componente especificando a composição correta de serviços;

Passo 3: Simultaneamente ao passo 2, faz-se necessário atualizar a ontologia do produto para que os demais agentes possam ter conhecimento desta nova combinação de serviços, que, de uma outra forma, representa uma nova *feature*, e dessa forma, a mesma possa ser utilizada pelos tutores;

Passo 4: Um agente que tem a responsabilidade de identificar a existência dessa nova *feature*, é o *Agente Verificador de Atualização do Produto*. Este agente tem a função natural de perceber quando um produto foi evoluído;

Passo 5: Em seguida, o *Agente Verificador de Atualização do Produto* realiza o seu trabalho de propagação, atualizando a ontologia da linha de produto. Assim, com esta atualização, torna-se possível que novos produtos possam utilizar esta nova *feature*.

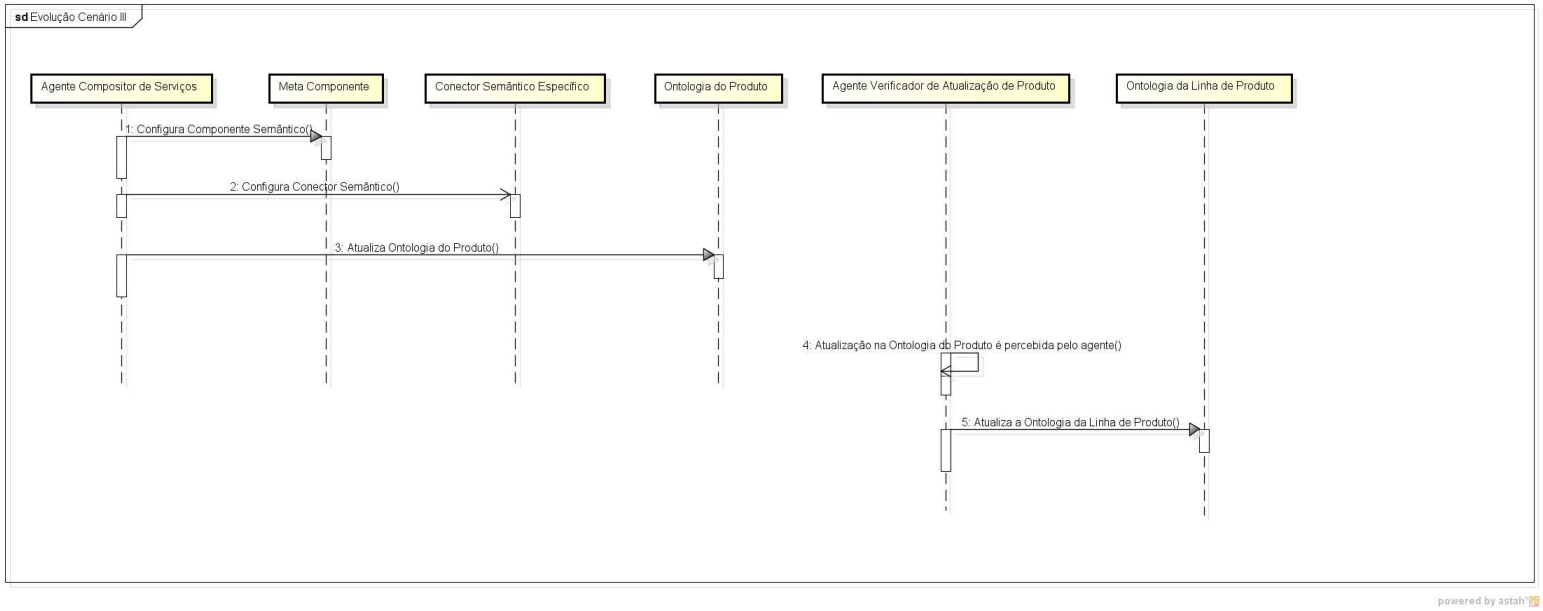


Figura 6.35: Diagrama de Sequencia do Cenário III

6.4 Discussão

Construir linhas de produtos de software e gerar seus produtos, respeitando os seus padrões e restrições, não é uma tarefa fácil. A especificação sem semântica de características de sistemas realiza a verificação de propriedades do sistema menos dependente do engenheiros de software e programadores. Desta forma, esta abordagem combinou ontologias ,agentes de software, serviços semânticos e linhas de produto, a fim de resolver, de forma combinada, as três questões de pesquisa desta tese. Conforme pode ser visto na Tabela 6.1, a solução desenvolvida nesta tese resolve as três questões de pesquisa, e isso pode ser constado com algumas evidências:

1. Produção em Larga Escala: Com o uso de Linhas de Produto foi possível desenvolver os artefatos de sistemas tutores inteligentes de uma maneira mais coordenada, no sentido de fornece um reuso sistematizado. Isso foi importante porque as possíveis configurações também são reutilizadas no processo de instanciação dos produtos, fazendo com que o seu processo de construção fosse otimizado.

2. Redução da Complexidade no Desenvolvimento em Larga Escala de STIs: Anotar semanticamente artefatos de software pode reduzir significativamente o esforço relacionado a engenharia de software, sobretudo na modelagem e implementação de aspectos cognitivos. Em relação a essa redução, pode-se notar na Figura 6.36 uma comparação com uma ferramenta consolidada na comunidade[1] para a construção deste tipo de sistema, assim, para o desenvolvimento de um sistema tutor, com a abor-

Tabela 6.1: Tabela Comparativa dos Trabalhos Relacionados - Questões de Pesquisa

Ambientes			
-	QUESTÃO 1	QUESTÃO 2	QUESTÃO 3
Ambiente com MAS e SOA	com restrições	sim	com restrições
ACOA	com restrições	sim	com restrições
Sem. e-Learning Fram.	com restrições	sim	não
Multitutor	com restrições	sim	não
Linhas de Produto Semântica	com restrições	com restrições	com restrições
Solução Desenvolvida Nesta Tese	sim	sim	sim

dagem desenvolvida nesta tese, que disponibilize uma hora de tutoria, foram gastos 20 horas de desenvolvimento. Enquanto que com uma ferramenta reconhecida na comunidade[1] gasta-se 250, vale ressaltar que a comparação de esforços mencionada se refere a dados de custo médio em sistema de complexidade semelhante, mas não foi medido no mesmo sistema. Isso acontece porque o conhecimento pode ser representado de uma maneira computacionalmente interpretável, com o uso de ontologias, fazendo com que o esforço de implementação seja reduzido.

3. Adaptação aos diferentes domínios: Com o uso de ontologias foi possível separar o conhecimento do software, assim, o software pode ser implementado de forma independente do domínio de sua aplicação. Dessa forma, é possível produzir STI para diferentes domínios com a mesma implementação. Sendo mais específico ao trabalho realizado nesta tese, o autor de um STI não precisa conhecer aspectos relacionados ao comportamento do sistema, portanto, somente é necessário conhecer aspectos sobre o domínio atual e as funcionalidades desejadas. Nesse sentido, o esforço e a complexidade para a construção são reduzidos a aspectos relacionados aos requisitos desejados.

4. Evolução Automatizada: A evolução automatizada foi possível com o uso de agentes inteligentes e serviços semânticos. Assim, sem a necessidade de reinicialização e sem a intervenção de profissionais especializado em computação, como programadores e engenheiros de software, é possível evoluir tanto a linha de produto como também os sistemas tutores inteligentes. Então, além de ser possível produzir em larga escala, é também possível promover uma adaptação em larga escala. Primeiramente porque as ontologias já proporcionam um compartilhamento natural no conhecimento que os sistemas tutores estarão manipulando, mas também pelo fato de que os demais artefatos são anotados semânticamente, viabilizando uma melhor interpretação por parte dos agentes de software. Dessa forma, os agentes de software poderão identificar o comportamento de cada sistema e analisar quais mudanças deverão ser implementadas.

6.4.1 Validação de Ontologias

Esta subseção visa apresentar como os agentes se comportaram na validação das ontologias usando o regras *SWRL*. Para isto, foram utilizadas duas ontologias de produto. Basicamente,

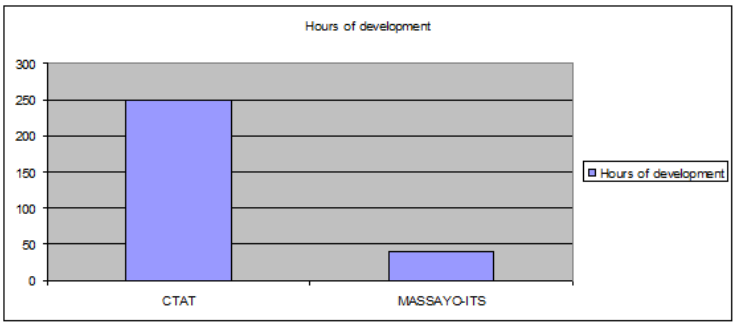


Figura 6.36: Horas de Desenvolvimento

ele irá descrever como a camada do motor de inferência funciona, e depois como está é utilizada depois da execução das regras. A implementação dos agentes segue a especificação descrita na Seção 5.4. Assim, a camada do motor de inferência é responsável pela execução regras *SWRL*. Esta camada utiliza a API Jess[2]. Quando o agente invoca esta camada para a execução de regra, está camada se adapta ao estado do agente.

Para isto, três agentes foram criados pelo Agente Gerente (*Manager Agent*): i) *SPLAgent_ITS*; ii) *ProductAgent_product1*; e iii) a *ProductAgent_product2*. Eles são responsáveis por: *ITS*, *product1*, *product2* , respectivamente. O *SPLAgent_ITS* recebe a ontologia a partir de seu Agente Gerente (*Manager Agent*). Depois que este agente invoca a camada mais abaixo. Na execução das regras relacionadas a SPL está previsto pela Camada da Máquina de Inferência ter retornos nulos. Portanto, o agente infere que a ontologia LPS é válida.

O segundo agente criado pelo Agente Gerente (*Manager Agent*) é *ProductAgent_product1*. Quando este agente invoca a máquina de inferência, houve duas regras que não retornaram nulo. A primeira foi a regra **Rule 6**, que seleciona todos os elementos selecionados cujo estado não foi selecionado. Isso ocorreu porque a *Feature Level of Knowledge* estava selecionada e o seu pai *(Learning Report)* não estava. A segunda regra é a Regra 5, que verifica se os recursos obrigatórios são selecionados. Neste caso, a feature *Test* é obrigatória, mas não foi selecionada. O *ProductAgent_product1* processa as informações e recomenda mudanças para o desenvolvedor, como pode ser observado na Figura 6.37.

Além disso, os agentes do tipo *Product Agent* ou emphSPL Agent podem combinar regras *SWRL*, a fim de superar as limitações das regras SWRL. O terceiro agente criado é *Pro-*

[2]Mais informações no site `http:\\www.jessrules.com\jess\index.html`

Figura 6.37: Mensagens para o Autor

ductAgent_product2, esse agente recebe a ontologia *product2* e começa a validação. Como já mencionado, o product2 tem todas as *features* alternativas com o estado atual eliminado.

A fim de encontrar inconsistências entre a Linha de Produto para Sistemas Tutores, o agente *ProductAgent_product2* usa a combinação de *Rule_9 Rule_8* e *Rule_10*. O *Rule_8* seleciona todas as *features* com o estado atual eliminados e que tenham o tipo de relação é igual a alternativa. Este agente armazena as features selecionadas pela *Rule_8* em sua lista *Feature-Alternative-Eliminated*.

A lista *ProductAgent_product2* cria dois auxiliares para guardar. O auxiliary1 armazena a lista de retorno de *Rule_9* e o *auxiliary2* armazena a lista de retorno de *Rule_10*. O *Rule_9* seleciona as características alternativas aos recursos da lista Matéria-Alternative-Eliminado. O *Rule_10* seleciona as características alternativas aos recursos da lista Matéria-Alternative-eliminados e que tenham estado atual equivale a eliminada. No final das execuções das regras, o *ProductAgent_product2* compara o conteúdo das listas de auxiliares. Se as listas forem iguais, o agente infere que todos os recursos alternativos com o estado atual eliminadas. Neste caso, o conteúdo das listas são iguais, porque o conteúdo das listas auxiliares são iguais a Passo a Passo e Geral.

Além disso, os agentes do tipo de *Product Agent* or *SPL Agent* têm um método que busca no diretório de regras para as regras SWRL novo, em outras palavras, o desenvolvedor pode escrever regras que o agente irá executar automaticamente. O processo de validação é repetido toda vez que a ontologia é alterada. Se a ontologia do agente é removida, esse agente vai terminar a sua execução.

6.4.2 Avaliação do Produto

No questionário aplicado para medir o nível de usabilidade do sistema tutor inteligente de programação, foi analisado, a partir do gráfico 6.38, que a usabilidade do sistema foi classi-

ficada como “regular” entre 37% dos alunos, como “bom” entre 36% destes e ótimo, entre 27%, o que deixa entender que a avaliação da clareza das funções das ferramentas feita pelos alunos, foi indispensável para perceber que o sistema está com um nível de entendimento dentro de padrões aceitáveis.

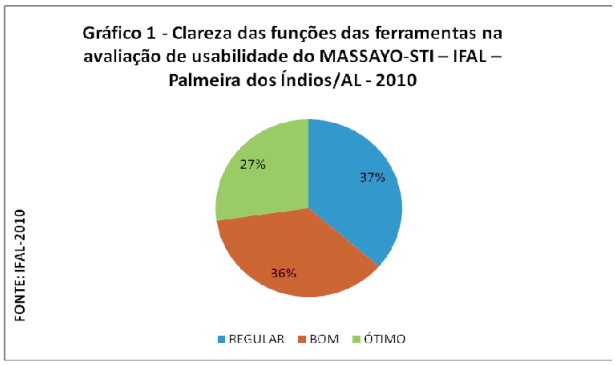


Figura 6.38: Modelos para a Instanciação de Produto

Diante da análise da usabilidade técnica e pedagógica do sistema, com base em gráficos, foi constatado que ele possibilita ao usuário executar tarefas de forma fácil e rápida, revelando mecanismos de fácil execução nas atividades. É adequado para usuários experientes e inexperientes e possibilita a compreensão e resolução de erros gerados pelo sistema, existe uma preocupação com a prevenção de erros. Eles são umas das principais frustrações, ineficiência e ineficácia durante a utilização do software, auxiliar o usuário a reconhecer, diagnosticar e recuperar é indispensável para o seu conforto, buscando sempre utilizar uma linguagem natural, ou seja, sem códigos, a fim de indicar o erro e sugerir uma solução imediata.

Segue alguns gráficos que exemplificam as afirmações acima:

O gráfico da Figura 6.39, mostra que 64% que o sistema possibilita ao usuário executar tarefas de forma fácil e rápida, 27% colocaram que pouco possibilita e 9% colocaram que não possibilita.

O gráfico da Figura 6.40, mostra o nível de adequação do sistema para os usuários experientes e não experientes. 25% classificaram como ótimo, 67% como bom e apenas 8% como regular.

O gráfico da Figura 6.41 mostra com 64%, que os alunos consideram que é possivel entender e resolver os erros gerados pelo sistema, mesmo 18% colocando que pouco possibilita

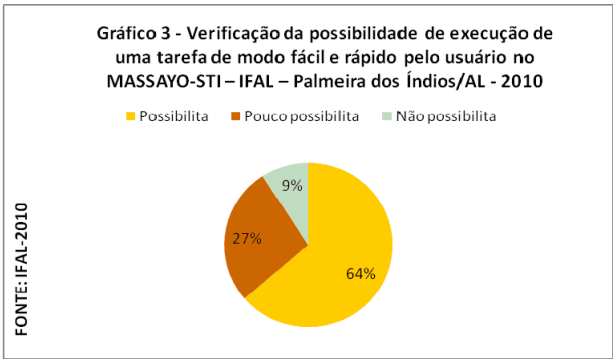


Figura 6.39: Modelos para a Instanciação de Produto

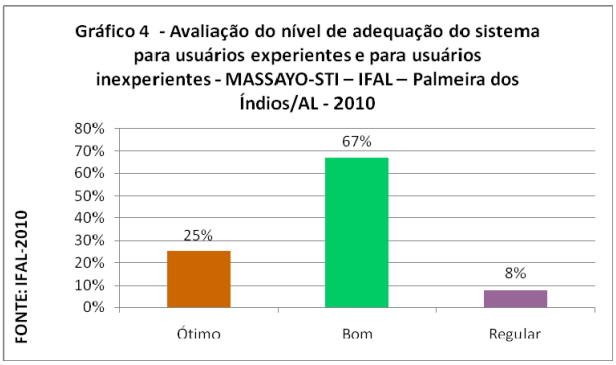


Figura 6.40: Modelos para a Instanciação de Produto

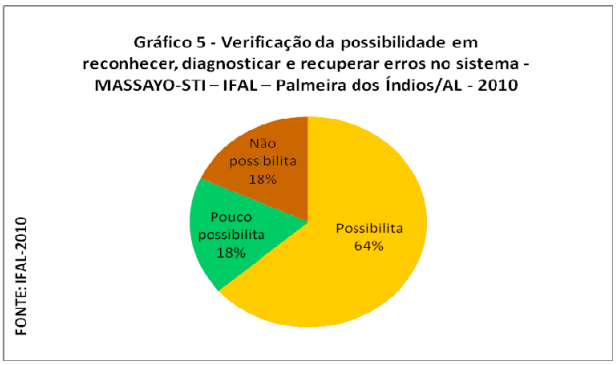


Figura 6.41: Modelos para a Instanciação de Produto

e os outros 18% afirmando que não possibilita.

Em relação a autonomia na aprendizagem, o sistema foi avaliado com 100%, como um possibilitador da autonomia, o que significa entender que o sistema tutor permite ao aluno desenvolver suas atividades de maneira independente. Mesmo entendendo que o conceito de autonomia é relativo, e se diferencia de um aluno para outro. Mas é fato que alguns programas permitem um maior exercício dessa independência nas realizações das atividades

de aprendizagem, pois admitem, com seu conjuto de objetivos e técnicas, que um número maior de alunos se adaptem sem se desmotivarem e permitem também que suplantem as dificuldades encontradas (vide gráfico da Figura 6.42):

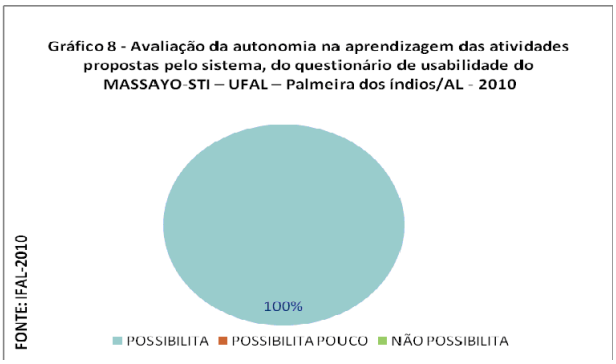


Figura 6.42: Modelos para a Instanciação de Produto

Um dos problemas atuais na educação é justamente a motivação para aprender, e é na busca por encontrar mecanismos que colaborem com tal prática que se busca estabelecer formas mais dinâmicas e atraentes de aprendizagem, em um mundo cada vez mais rodeado por mecanismos tecnológicos, faz-se necessário adaptar tais mecanismos aos processos educativos. O papel da educação atual é propor o gosto pelo aprender, a curiosidade intelectual.

O gráfico da Figura 6.43, classifica, com 82%, como bom o nível de motivação gerado pelo sistema tutor, fazendo assim entender que é um ambiente de apredizagem que possibilita ao aluno se sentir motivado em suas atividades e estudos.

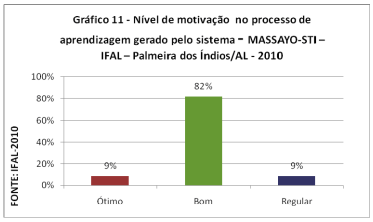


Figura 6.43: Modelos para a Instanciação de Produto

Diante desses aspectos entende-se que o tutor, sugere como objetivo, auxiliar o usuário (aluno) através de um processo interativo e dinâmico, com grande habilidade em uma composição de algumas disciplinas como a psicologia, as ciências do conhecimento e a inteligência artificial, para responder as necessidades dos usuários, em uma proposta de ensinar não

só o conteúdo, mas na busca de formas mais agradáveis de tal processo, a fim de ensinar a aprender.

6.4.3 Avaliação da Linha de Produto

A linha de produto foi avaliada sob dois conjuntos de métricas. A primeira baseada em Compreensibilidade, Analisabilidade e Mutabilidade, que foram propostas por Bagueri[6] e o segundo conjunto foi proposto por Zhang[97], para medir a similaridade, a variabilidade e a reutilização da arquitetura de linha de produtos. Assim, a primeira avaliação está descrita na Subseção 6.4.3 e a segunda avaliação está descrita na Subseção 6.4.3, conforme a seguir:

Avaliação Baseada em Compreensibilidade, Analisabilidade e Mutabilidade

Nesta Subseção está demonstrada a avaliação da linha de produto baseada em compreensibilidade, analisabilidade e mutabilidade[6]. Para isto, foi tomado como base as métricas propostas por [6]. Como um primeiro passo, foi calculado os índices propostos pelo mesmo trabalho (demonstrados na Tabela 6.2). Depois de calcular estes índices, fez-se necessário avaliá-los comparando com os mesmos índices calculados em outras linhas de produto já avaliadas por especialistas (conforme os dados expostos na Figura 6.44). Além disso, o resultado destas avaliações foram colocados em uma escala de 1 a 7 (Figura 6.45), onde o 1 representa que uma das métricas (Compreensibilidade ou Analisabilidade ou Mutabilidade) é extremamente difícil e o 7 representa que uma destas mesmas métricas é extremamente fácil.

Feature Model	NF	Ntop	Nleaf	CC	CTC	RoV	CoC	FoC	NVC	DT
DELL Laptop/ Notebook Computers	48	8	37	110	0,804	3	1,889	0,021	858	9
Inventory	37	5	28	6	0,37	4	1,135	0,162	2028256	4
Sienna	38	4	34	8	0,263	2,642	1,184	0,131	2520	5
HIS	67	7	28	4	0,119	2,129	1,064	0,149	9400	7
Documentation Generation	44	9	32	8	0,295	3,045	1,139	0,098	357600000	8
Thread	44	1	36	0	0	2,388	0,977	0,34	80658	13
Smart Home	35	6	25	0	0	2,833	0,971	0,171	1048576	9
Arcade Game	61	2	47	36	0,557	4,285	1,24	0,081	8000400000	8
Model Transformation	88	8	55	0	0	2,636	0,988	0,124	1.85600004E+009	9
Search Engine	14	4	10	2	0,185	1,35	1,071	0,357	136	4
Text Editor	18	2	10	0	0	2,135	0,944	0,444	990	4
Web Portal	43	5	28	6	0,255	2,8	1,116	0,899	2120800	5
Electronic Shopping	287	3	193	21	0,118	1,01	1,009	0,383		13
Bigata	27	5	20	2	0,148	3,714	1,037	0,222	1152	5
TOTAL	849	69	558	201	3,114	44,395	17,624	3,359	1.65498E+009	88
Base	60,64286	4,5	39,85714	14,35714	0,222429	3,171071	1,259429	0,239929	1,17845E+012	6,285714

Extremely Difficult (1)	Very Difficult (2)	A Bit Difficult (3)	Neither Difficult Nor Easy (4)	Quite Easy (5)	Very Easy (6)	Extremely Easy (7)

Figura 6.45: Escala para avaliação de especialistas[6]

Assim, foi possível comparar a linha de produto, com as outras linhas de produtos, pois, cada uma delas já foi avaliada, conforme Figura 6.46. O Resultado desta avaliação está numa escala previamente estabelecida, conforme as escalas da Figura 6.45.

Feature Model	Analysability	Understandability	Changeability
DELL Laptop/Notebook Computers			
median	2	3	2
Inventory			
median	3	5	4
Sienna			
median	5	4	4
HIS			
median	4	4	4
Documentation Generation			
median	2	4	3
Thread			
median	4	5	7
Smart Home			
median	3	5	5
Arcade Game			
median	2	3	2
Model Transformation			
median	2	3	5
Search Engine			
median	6	6	5
Text Editor			
median	5	5	6
Web Portal			
median	3	5	4
Electronic Shopping			
median	2	4	4
Bicycle			
median	5	6	5

Figura 6.46: Resultado da avaliação de especialistas[6]

Além disso, para fazer uma avaliação de forma mais acurada, [6] oferece mecanismos para avaliar as relações entre índices, de forma a oferecer subsídios para quantificar mais adequadamente as métricas propostas pelo mesmo, conforme as relações expostas nas Figuras 6.47 e 6.48. Por exemplo, pode-se notar que o *NF* (número de recursos) possui uma alta correlação com a métrica *NLeaf* (número de folhas) e, como ambos possuem alta correlação negativa com analisabilidade e compreensibilidade, podemos inferir que quanto maior o número de recursos, maior será o numero de folhas na árvore, e mais complexo será o modelo de *feature*, em termos de analisabilidade e compreensibilidade. Portanto, o *NLeaf* e *NF* podem ser usado como bons indicadores para analisabilidade e compreensibilidade.

	NF	DT	NTop	NLeaf	CC	CTC	RoV	CoC	FoC	NVC
NF		**0.88**	0	**0.95**	0.41	0	-0.01	0.24	-0.53	**0.7**
p value		0.01	0.99	0	0.14	1	0.98	0.41	0.05	0.01
DT			-0.32	**0.84**	0.01	**-0.27**	-0.3	-0.1	-0.2	**0.61**
p value			0.27	0.03	0.96	0.36	0.3	0.73	0.49	0.02
NTop				0.01	-0.06	0.03	0.11	0.01	-0.18	-0.21
p value				0.96	0.83	0.87	0.71	0.98	0.54	0.46
NLeaf					**0.84**	0.17	0.22	0.39	**-0.59**	**0.75**
p value					0.05	0.55	**0.44**	0.17	0.03	0
CC						**0.8**	**0.65**	**0.91**	**-0.67**	0.23
p value						0	0.01	0	0.01	0.42
CTC							0.78	**0.25**	**-0.65**	-0.1
p value								0	0.01	0.73
RoV								**0.69**	-0.47	0.09
p value								0.01	0.09	0.75
CoC									**-0.74**	0.11
p value										0.71
FoC										0.98
p value										0.49

Figura 6.47: Relação entre os índices propostos[6]

	NF	DT	NTop	NLeaf	CC	CTC	RoV	CoC	FoC	NVC
Analysability	-0.75	-0.27	0.01	**-0.88**	-0.48	-0.21	-0.39	-0.35	0.49	**-0.77**
p value	0.00	0.33	0.96	0.00	0.07	0.45	0.15	0.20	0.07	0.00
Understandability	-0.81	-0.22	-0.07	**-0.82**	-0.53	-0.26	-0.12	-0.45	0.74	-0.48
p value	0.00	0.25	0.79	0.00	0.04	0.38	0.68	0.09	0.00	0.07
Changeability	-0.46	0.04	-0.07	-0.60	**-0.92**	**-0.80**	**-0.65**	**-0.88**	0.73	-0.32
p value	0.09	0.88	0.78	0.02	0.00	0.00	0.01	0.00	0.00	0.28

Figura 6.48: Relação entre os índices propostos e as bases da manutenção[6]

Por outro lado, foi identificado que o *CC* (Complexidade ciclomática) apresenta um correlação com *CoC* e *CTC*, e possuem alta correlação negativa com a mutabilidade. Pelo mesmo motivo acima, esses três indicadores podem ser considerados bons indicadores para a mutabilidade.

Dessa forma, a partir dos dados da tabela 6.2, e baseando-se nos dados apresentados no trabalho do Dragan [6], pode-se afirmar que a linha de produto apresentada para sistemas tutores inteligentes apresenta as seguintes características:

- Por possuir um NF (número de features) abaixo da média base, e por consequente, apresentar um Nleaf (número de folhas) também baixo, e como dito anteriormente, ambas as métricas possuem correlação negativa com a analisabilidade e compreensibilidade, então o modelo de feature apresenta baixa complexidade, no que diz respeito à analisabilidade e compreensibilidade, isto é, é fácil compreender o modelo e encontrar deficiências nele, caso elas existam.

- Ao analisarmos os índices de CC, CoC e CTC foi visto que tanto o CC quanto o CTC apresentam-se abaixo da média, porém o índice CoC está pouco acima da média. Devido à correlação negativa entre essas métricas e a mutabilidade, pode-se afirmar que o modelo apresentado encontra-se no nível “Nem difícil nem fácil”, com relação à mutabilidade.

- O fato das medidasNTope DT estarem próximos da média base não influência na manutenbilidade, uma vez que sua correlação é pequena, se comparada as outras métricas.

- Ao se analisar a medida RoV, pode-se notar que seus resultados são compatíveis com os resultados da NLeaf e Nf, apresentando que o modelo possui baixa complexidade, em relação à compreensibilidade e analisabilidade.

- A medida FoC, que analisa a flexibilidade de configuração do modelo, apresenta-se bastante elevada, o que confirma ainda mais o fato do modelo ser compreensível, uma vez que essa medida está correlacionada com compreensibilidade.

- Por último, foi analisado a medida NVC, que está abaixo da média. Mais um fato que mostra que o nível de analisabilidade do modelo é alto.

Tabela 6.2: Calculo dos índices que avaliam linhas de produto

Métricas	**Valores**
Número de Features (NF)	36
Número de principais Features (Ntop)	6
Número de features folha (Nleaf)	23
Profundidade da árvore (DT)	5
Complexidade ciclomática (CC)	1
Restrições Cross-Tree(CTC)	0,027
Proporção de variabilidade (RoV)	0,972
Coeficiente de Conectividade/Densidade (CoC)	1,36
Flexibilidade de configuração (FoC)	0,527
Número de configurações válidas (NVC)	524288

Em suma, tomando como base os resultados apresentados, pode-se afirmar que a linha de produto para sistemas tutores, aqui apresentada, é completa, no que diz respeito à compreensibilidade, isto é, a possibilidade e/ou probabilidade do modelo ser compreendido é extremamente alta. A analisabilidade do modelo é simples, o que significa que é fácil a detecção de erros no modelo conceitual do sistema, caso eles existam. Por outro lado, por apresentar um nível de mutabilidade "Nem difícil nem fácil", o modelo tende a ficar mais estável, sem evolução significativa, uma vez que as mudanças no modelo são difíceis de serem implantadas.

Avaliação baseada em Similaridade

Nesta subseção está descrita a avaliação baseada em métricas de similaridade, propostos por [97], para medir a similaridade, a variabilidade e a reutilização da arquitetura de linha de produtos. Essas métricas ajudam a analisar e promover a qualidade da arquitetura da linha de produto. Nesse sentido, as linhas de produto de software contém um conjunto de sistemas de software em um domínio especial. As arquiteturas dos membros da linha devem ser semelhantes, o que é fundamental para a reutilização. Ao medir similaridade, pode-se aperfeiçoar o alcance da linha de produto. Para isto, foi proposto, primeiramente, o Structure

Similarity Coefficient (SSC), definido como:

$$SSC = \frac{|Cc|}{|Cc| + |Cv|},$$

Onde $|Cc|$ é o numero de componentes comuns na linha de produto e $|Cv|$ é o número de componentes variáveis na linha de produto.

Variabilidade é a característica fundamental de uma linha de produto, e é importante para satisfazer as peculiaridades dos membros da linha de produto. Assim, as métricas de variabilidade ajudam a melhorar a concepção da linha, com relação à expansibilidade e analisabilidade. Para isto, foram propostas algumas métricas, descritas abaixo:

- ***Variability Point (VP)***: É o numero de pontos de variabilidade na interface da linha de produto. Note que quanto maior o VP, mais arquiteturas pode ser derivadas da linha e maior será sua complexidade e maior será a dificuldade de *design*.

- ***Strong Coupling Coefficient (SCC)***: É proposto para medir o acoplamento entre as variabilidades da linha. É calculada com a seguinte fórmula:

  $$SCC = 1 - \frac{|IVP|}{|VP|},$$

  Onde $|IVP|$ é o numero de pontos de variáveis independentes.

- ***Weak Coupling Coefficient (WCC)***: É definido como medida de acoplamento fraco. Note que um fraco acoplamento dificulta no design. É calculado através da seguinte formula:

  $$WCC = \frac{|CVP|}{|VP|},$$

  Onde $|CVP|$ é o numero de pontos de variabilidade de acoplamento fraco.

- ***Structure Variability Coefficient (SVC)***: É definido como medida da estrutura de variabilidade de uma linha de produto. Calculada a partir da seguinte formula:

  $$SVC = \frac{|Cv|}{|Cc| + |Cv|},$$

Portanto, no primeiro momento, foi preciso calcular os valores das métricas. E estes resultados estão detalhados na tabela abaixo:

Métricas	Valor
$\|Cc\|$	15
$\|Cv\|$	21
SSC	0,417
VP	20
$\|IVP\|$	11
SCC	0,45
$\|CVP\|$	9
WCC	0,45
SVC	0,584

Dessa forma, algumas constatações puderam ser observadas:

Por possuir poucos componentes comuns, e mais componentes variáveis, implicando no *SSC* baixo, concluímos que a linha de produto apresenta pouca similaridade, perdendo benefícios de reuso.

Notamos que a linha apresenta um *VP* razoável, se comparamos ao numero total de *features* 36, o que implica que pode ser derivado um grande numero de arquiteturas da linha, porém apresenta um nível de complexidade razoável.

Pelo *SCC* apresentar-se em nível baixo, concluímos que o acoplamento entre os pontos de variabilidade estão em níveis baixos, possuindo assim, pouca dificuldade ao instanciar um *PLA*.

Com esse, *WCC*, concluímos que a linha de produto não apresenta problemas com relação ao design, uma vez que o valor é baixo.

Por possuir um *SVC* alto, podemos concluir que a linha de produto apresenta uma alta compreensibilidade e mutabilidade.

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho se insere no contexto da pesquisa atual na área de Inteligência Artificial em Educação (IAEd), buscando-se produção efetiva e em larga escala de Sistemas Tutores Inteligentes baseados em agentes e seguindo um enfoque semântico. Nesse sentido, seus objetivos, tal como declarados no Capítulo 1, foram atingidos, tendo a seguir resumidos os seus principais resultados alcançados, destacando-se as suas principais contribuições para a área de pesquisa mencionada acima. Além disso, algumas de suas limitações são discutidas, culminando com uma lista de sugestões para estendê-lo em trabalhos futuros.

Portanto, esta tese apresentou resultados que são soluções para as questões de pesquisa, conforme discutidas no Capítulo 1, relacionadas à produção em larga escala de sistemas tutores inteligentes. Estes problemas acontecem por conta da complexidade que está embutida em tais sistemas, produzindo um custo alto em sua construção. Nesse sentido, a pesquisa realizada nesta tese tratou especificamente este problema utilizando linha de produto de software sob uma perspectiva da web semântica, conduzindo às contribuições para as áreas de IAEd e Engenharia de Software, tais como resumidas a seguir:

Produção em Larga Escala Em um primeiro momento, a utilização de linha de produto de forma tradicional não se mostrou interessante para o produto central desta tese: Sistema Tutor Inteligente. Isso aconteceu, primeiramente, pela necessidade de adaptação destes produtos entre os diferentes domínios de conhecimento. Pois, caso contrário, se houver linhas de produto diferentes para cada domínio de conhecimento, o custo de produção dos tutores poderá ser bastante elevado, principalmente por conta de sua complexidade. Assim, o uso de ontologias se mostrou viável para este propósito. Isso

é justificado por alguns motivos: (i) As ontologias têm um grande poder de expressividade, e aliado a essa característica, ainda existe a possibilidade de verificação de consistência de forma automatizada. Por isso, o conhecimento relacionado aos tutores pode ser movido da camada de software dos tutores, para uma camada de conhecimento baseada nas ontologias. Assim, a implementação dos tutores pode ser voltada para o conhecimento abstrato de um tutor, de forma que o domínio de conhecimento se torna indiferente. (ii) Além disso, alguns artefatos também puderam ser anotados com esta tecnologia, como componentes e conectores. Com isso, a linha também pode ser flexível para os diferentes domínios de conhecimento, dado que os artefatos responsáveis pela configuração dos produtos "não precisam ter conhecimento" do domínio que o mesmo está tratando.

Adaptação aos Diferentes Domínios: Tanto a linha de produto, quanto os produtos se mostraram possíveis de serem adaptáveis a diferentes domínios de conhecimento. Isso foi possível com o uso da web semântica porque os artefatos podem ser anotados semanticamente, e de forma padronizada, de tal forma que a busca por recursos pode ser feita de maneira mais precisa. Além disso, com o suporte da tecnologia de agentes, é possível direcionar a configuração dos produtos de forma individualizada, respeitando possíveis individualidades de respectivos contextos. Por outro lado, os próprios produtos têm a capacidade por si próprio de se adaptar a um domínio de conhecimento, por conta de padronização de recursos educacionais.

Evolução Automatizada: Do ponto de vista de evolução, a web semântica também se tornou uma possível solução, através do uso de agentes inteligentes e serviços semânticos. Isso é possível porque serviços semânticos podem ser compostos para formar alguma funcionalidade previamente estabelecida como uma *feature* em algum ponto de variabilidade. Os agentes tanto podem descobrir uma determinada combinação de serviços semânticos, como podem atualizar a linha, e selecionar quais produtos podem utilizar a nova configuração de serviços. Isso é importante no sentido de reduzir a demanda por engenheiros de softwares, programadores, e especialistas em educação.

Além disso, se considerarmos o fato de ser possível desenvolver STIs com funcionalidades distintas para um mesmo domínio de conhecimento, este trabalho pode ser visto como

uma solução para construção de famílias de SPL. Assim, espera-se ter contribuído com as seguintes comunidades:

- Inteligência Artificial e Educação: Construção em larga escala de sistemas tutores inteligentes, de modo personalizado, de tal forma que é abstraido a configuração dos produtos para autores sem conhecimento especializado de computação;

- Engenharia de Software: na proposição de ontologias que permitiram a especificação de sistemas baseados em agentes e na geração dos agentes de software e a necessidade de implementação. Além disso, na integração de ontologias, agentes e serviços semânticos para reduzir o tempo de manutenção das aplicações;

Foram também alcançados alguns outros indicadores como parte do processo de pesquisa associado à realização desta tese. Tais indicadores são: i) Organização de evento[1] de Web Semântica e Educação do País; ii) Participação em comitê de conferência especializada[2]; iii) publicação em conferências nacionais [84] e internacionais [83] v) Aprovação e coordenação de projeto de pesquisa financiado pela FAPEAL.

A pesquisa abordada nesta tese possui vários trabalhos futuros, sendo tais extensões propostas a seguir.

Estratégias Pedagógicas Como trabalhos futuros relacionados as estratégias pedagógicas, sugere-se principalmente:

- Implementação de outros componentes de estratégias pedagógicas baseados nas diversas teorias de aprendizagem;
- Analisar estes outros componentes, e analisar se o modelo para estratégias pedagógica é suficiente para distingui-los, principalmente sob uma perspectiva de automação do seu processo de escolha.

Aprendizagem de Máquina Ajuste de técnicas de aprendizagem de máquina para serem embutidas no agentes inteligentes, para que os mesmos possam ser utilizados efetivamente na tarefa de escolha dos componentes de cada tutor:

[1] III Brazilian Workshop on Semantic Web and Education, em conjunto com o SBIE 2010.

[2] *ACM Symposium On Applied Computing 2011 (Track on Intelligent, Interactive and Innovative Learning environments)*

- Foi apresentado a capacidade de agentes tomarem decisões de quais componentes são mais apropriados para cada perfil de aluno;
- No entanto, faz necessário um conhecimento prévio de como esta mudança podem ocorrer;
- Nesse sentido, pensando em uma distribuição irrestrita de componentes para a linha, é interessante que seja provido técnicas que identifiquem automaticamente quando estes componentes são mais apropriados.

Integração com Ambientes Virtuais de Aprendizagem Tradicionais Os produtos gerados pela linha de produto proposta podem ser acoplados a ambientes virtuais tradicionais, tais como: O Moodle e o SAKAI. Isso é importante porque os sistemas tutores poderiam se beneficiar de outros mecanismos de interação, e portanto, forcer um ensino mais sofisticado.

Maior Robustez Apesar das vantagens comentadas no trabalho, o uso de serviços web semânticos pode trazer consequências indesejadas, principalmente:

- Problemas de desempenho: Pretende-se uma revisão arquitetural da linha de produto proposta nesta tese, de tal forma que os serviços utilizados possam ser executados de forma mais escalável. Isso acontece porque os serviços semânticos podem estar sobrecarregados, e por conta disso, pode haver uma necessidade de se projetar mecanismos computacionais no sentido de promover uma distribuição de carga.
- Dificuldade técnica na criação dos serviços: Além disso, o próprio processo de criação dos serviços semânticos precisa ser melhor elaborado, para que este tipo de serviço possa ser amplamente desenvolvido.

Bibliografia

[1] Vincent Aleven, Bruce M. McLaren, Jonathan Sewall, and Kenneth R. Koedinger. The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, and T. W. Chan, editors, *International Conference on Intelligent Tutoring Systems*, pages 61–70. Springer Verlag, 2006.

[2] Vincent Aleven, Bruce M. McLaren, Jonathan Sewall, and Kenneth R. Koedinger. The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In *Intelligent Tutoring Systems*, pages 61–70, 2006.

[3] Lora Aroyo, Akiko Inaba, Larisa Soldatova, and Riichiro Mizoguchi. Ease: Evolutional authoring support environment. In *Intelligent Tutoring Systems*, pages 140–149, 2004.

[4] Colin Atkinson, Joachim Bayer, and Dirk Muthig. Component-based product line development: the kobra approach. In *Proceedings of the first conference on Software product lines : experience and research directions*, pages 289–309, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[5] Faiçal Azouaou and Cyrille Desmoulins. Using and modeling context with ontology in e-learning: the case of teacher's personal annotation. In *International Workshop on Applications of Semantic Web Technologies for E-Learning (SW-EL), In Adaptive Hypermedia and Adaptive Web-Based Systems, Dublin, Ireland*, 2006.

[6] Ebrahim Bagueri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 2010.

[7] Heitor Barros, Alan Pedro, Ig Ibert Bittencourt, Evandro Costa, Olavo Holanda, and Leandro Sales. Steps, techniques, and technologies for the development of intelligent

applications based on semantic web services: A case study in e-learning systems. *International Scientific Journal Engineering Applications of Artificial Intelligence*, 2011.

[8] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2 edition, April 2003.

[9] J. Beck, I. Arroyo, B. P. Woolf, and C. R. Beal. Affecting self-confi dence with an its. In *Ninth International Conference on Artifi cial Intelligence in Education*, 1999.

[10] Joseph E. Beck, Beverly Park Woolf, and Carole R. Beal. Advisor: A machine learning architecture for intelligent tutor construction. In *Proceedings of the National Conference on Artificial Intelligence*, pages 552–557, 2000.

[11] I. I. Bittencourt, S. Isotani, E. Costa, and R. Mizoguchi. Research directions on semantic web and education. *Journal Scientia - interdisciplinary studies in Computer Science*, 19(1):59–66, 2008.

[12] Ig Ibert Bittencourt, Evandro Costa, Marlos Silva, and Elvys Soares. A computational model for developing semantic web-based educational systems. *Know.-Based Syst.*, 22(4):302–315, 2009.

[13] Ig Ibert Bittencourt, Camila Nunes, Camila Bezerra, Evandro de Barros Costa, Rômulo Nunes de Oliveira, Marcos Costa, Marcos Tadeu, and Alan Pedro da Silva. Ontologia para construção de ambientes interativos de aprendizagem. In *Simpósio Brasileiro de Informática na Educação*, pages 559–568, 2006.

[14] Benjamin S. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(8):4–16, 1984.

[15] Jesús G. Boticario, Luis Castillo Vidal, Ramón Fabregat Gesa, Manuel Ortega, Daniel Borrajo, and Eva Onaindia de la Rivaherrera. Adaptation based on machine learning, user modelling and planning for complex user-oriented tasks: Adaptaplan (tin2005-08945-c06-00). In *Jornadas de Seguimiento de Proyectos. Programa Nacional de Tecnologías Informáticas*, 2007.

[16] Marco Brambilla, Irene Celino, Stefano Ceri, Dario Cerizza, Emanuele Della Valle, and Federico Michele Facca. A software engineering approach to design and development of semantic web service applications. In *International Semantic Web Conference*, pages 172–186, 2006.

[17] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: An architecture for storing and querying rdf data and schema information. In *Semantics for the WWW*. MIT Press, 2001.

[18] Ivo Calado, Heitor Barros, and Ig Ibert Bittencourt. An approach for semantic web services automatic discovery and composition with similarity metrics. *SAC*, pages 694–695, 2009.

[19] Francesca Carmagnola, Federica Cena, Cristina Gena, and Ilaria Torre. A multidimensional semantic framework for adaptive hypermedia systems. In *IJCAI*, pages 1551–1552, 2005.

[20] John Cheesman and John Daniels. *UML components: a simple process for specifying component-based software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[21] Feng Chen, Qianxiang Wang, Hong Mei, and Fuqing Yang. An architecture-based approach for component-oriented development. *26th Annual International Computer Software and Applications Conference*, August 2002.

[22] W. Chen and R. Mizoguchi. Leaner model ontology and leaner model agent. In *Cognitive Support for Learning - Imagining the Unknown,*, 2004.

[23] V. I. Chepegin, L. Aroyo, and P. D. Bra. Ontology-driven user modeling for modular user adaptive systems. In *LWA*, pages 17–19, 2004.

[24] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[25] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, January 2000.

[26] Benedito Luiz Correia, Dante Henrique Moura, Denio Rebello Arantes, Jane Paiva, Maria da Conceição V. P. Oliveira, Marilise Braivante, Simone Valdete dos Santos, and Tânia Midian Freitas de Souza. *Programa Nacional De Integração Da Educação Profissional Com A Educação Básica Na Modalidade De Educação De Jovens e Adultos*. Ministério da Educação - Secretaria de Educação Profissional e Tecnológica, 2007.

[27] E. B. Costa. *Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura Multi-Agentes*. PhD thesis, Universidade Federal da Paraíba, Campina Grande, 1997.

[28] Evandro B. Costa. *Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura Multi-Agentes*. Tese de doutorado, Universidade Federal da Paraíba, Campina Grande, 1997.

[29] Mateus B. Costa, Rodolfo F. Resende, Eduardo F. Nakamura, and Marcelo V. Segatto. Software frameworks for information systems integration based on web services. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 777–782, New York, NY, USA, 2008. ACM.

[30] Alejandro Canales Cruz, Ruben Peredo Valderrama, Oscar Fabela Cano, and Humberto Sossa A. Architecture for development of wbes based on components and agents. In *15th International Conference on Computing, CIC '06*, pages 223–228, 2006.

[31] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[32] Alan Pedro da Silva, Evandro Costa, Ig Ibert Bittencourt, Patrick H. S. Brito, Olavo Holanda, Jean Melo, Diego Dermeval, and Márcio Ribeiro. Ontology-based software product line for building semantic web applications. In *In 1st International Workshop on Knowledge-Oriented Product Line Engineering (KOPLE 2010) in conjunction with ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH 2010)*, October 2010.

[33] S.M. Davis. *Future Perfect*. Addison-Wesley, 1987.

[34] Vladan Devedzic. Key issues in next-generation web based education. *IEEE Transaction on Education*, 3(33):339–349, 2003.

[35] Vladan Devedzic. *Semantic Web and Education*. Springer, 1 edition, 2006.

[36] Pierre Dillenboug and John Self. A framework for learner modelling. *Interactive Learning Environments*, 2(111-137), 1992.

[37] Marius Dragomiroiu, Marian Ventuneac, Ioan Salomie, and Tom Coffey. Application framework development for virtual learning environments. In *25th Int. Conf. lnformation Technology lnterfaces IT, Cavtat, Croatia*, 2003.

[38] B. du Boulay and R Luckin. Modeling human teaching tactics in a computer tutor. In *International Journal of Artificial Intelligence in Education*, pages 235–256, 2001.

[39] Eman El-Sheikh and Jon Sticklen. A framework for developing intelligent tutoring systems incorporating reusability. In *IEA/AIE '98: Proceedings of the 11th international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 558–567, London, UK, 1998. Springer-Verlag.

[40] Larbi Esmahi and Fuhua Lin. *Designing Distributed Learning Environment with Intelligent Software Agents*, chapter A Multiagent Framework for an Adaptive E-Learning System, pages 218–241. Idea Group, 2005.

[41] L. A. Gayard, C. M. F. Rubira, and P. A. C. Guerra. COSMOS: a component system model for software architecture. Technical report, Universidade Estudual de Campinas, 2008.

[42] Rosario Girardi and Adriana Leite. A knowledge-based tool for multi-agent domain engineering. *Know.-Based Syst.*, 21(7):604–611, 2008.

[43] Hassan Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[44] Gomez-Perez Gomez-Perez, Rafael González-Cabero, and Manuel Lama. Ode sws: A framework for designing and composing semantic web services. *IEEE Intelligent Systems*, 19(4):24–31, 2004.

[45] Bradley Goodman, Marty Geier, Lisa Haverty, Frank Linton, , and Robert McCready. A framework for asynchronous collaborative learning and problem solving. In *In the Proceedings of the 10th International Conference on Artificial Intelligence in Education, AIED*, November 2001.

[46] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, page 45, Washington, DC, USA, 2001. IEEE Computer Society.

[47] Andreas Helferich, Georg Herzwurm, Stefan Jesse, and Martin Mikusz. Software product lines, service-oriented architecture and frameworks: worlds apart or ideal partners? In *Proceedings of the 2nd international conference on Trends in enterprise application architecture*, TEAA'06, pages 187–201, Berlin, Heidelberg, 2007. Springer-Verlag.

[48] Weihong Huang, David Webster, Dawn Wood, and Tanko Ishaya. An intelligent semantic e-learning framework using context-aware semantic web technologies. *British Journal of Educational Technology*, 37(3):351–373, May 2006.

[49] Nicholas R. Jennings and Michael Wooldridge. Agent-oriented software engineering. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Minneapolis, Minnesota USA*, 2000.

[50] Jelena Jovanovic, Dragan Gasevic, and Vladan Devedzic. Tangram for personalized learning using the semantic web technologies. *Journal of Emerging Technologies in Web Intelligence*, 1(1):6–21, 2009.

[51] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[52] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[53] Barbara Kieslinger and Bernd Simon. Elena project: Creating a smart space for learning. Technical report, Information Society Technology, 2005.

[54] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM.

[55] E. J. R. Koper. Modeling units of study from a pedagogical perspective - the pedagogical meta-model behind eml. In *Open University of The Netherlands*, November 2001.

[56] V. Kumar, J. Shakya, C. Groeneboer, and S. Chu. Toward an ontology of teaching strategies. In *Proceedings of the ITS04 Workshop on Modelling Human Teaching Tactics and Strategies*, pages 71–80, 2004.

[57] Fuhua Lin, P. Holt, S. Leung, M. Hogeboom, and Y. Cao. A multi-agent and service-oriented architecture for developing integrated and intelligent web-based education systems. In *In Proc. International Workshop on Semantic Web Technologies for E-leanring, Maceios, Alagoas, Brazil*, pages 11–20, 2005.

[58] Fuhua Lin, Peter Holt, Steve Leung, and Qin Li. A multiagent and service-oriented architecture for developing adaptive e-learning systems. *Int. J. Cont. Engineering Education and Lifelong Learning*, 16(1/2):77–91, 2006.

[59] LORNET. Lornet: Portals and services for knowledge management and learning on the semantic web. Disponível em: http://www.lornet.org/, 2008. Acesso em 04 de Dez. de 2008.

[60] Carlos J. P. Lucena, Hugo Fuks, Ruy Milidiu, Carlos Laufer, Marcelo Blois, Ricardo Choren, Viviane Torres, Fabio Ferraz, Gustavo Robichez, and Leandro Daflon.

Aulanet: Ajudando professores a fazerem seu dever de casa. In *XXVI SEMISH - Seminário Integrado de Software e Hardware - Sociedade Brasileira de Computação*, pages 105–117, 1999.

[61] David Millard, Karl Doody, Hugh Davis, Yvonne Howard, Lester Gilbert, Feng Tao, and Gary Will. (semantic web) services for e-learning. In *2nd International ELeGI Conference on Advanced Technology for Enhanced Learning, Barcelona, Spain*, Oct 2006.

[62] Antonija Mitrovic and Vladan Devedzic. A model of multitutor ontology-based learning environments. *Int. J. Cont. Engineering Education and Lifelong Learning*, 14(3):229–245, 2004.

[63] Bardia Mohabbati, Nima Kaviani, and Dragan Gasevic. Semantic variability modeling for multi-staged service composition. In *In proceedings of the 3rd international workshop on Service-Oriented Architectures and Software Product Lines*, 2009.

[64] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE Softw.*, 14(1):43–52, 1997.

[65] Pedro J. Mu noz Merino and Carlos Delgado Kloos. An architecture for combining semantic web techniques with intelligent tutoring systems. In *ITS '08: Proceedings of the 9th international conference on Intelligent Tutoring Systems*, pages 540–550, Berlin, Heidelberg, 2008. Springer-Verlag.

[66] Lydia Silva Muñoz and José Palazzo Moreira de Oliveira. Applying semantic web technologies to achieve personalization and reuse of content in educational adaptive hypermedia systems. In *International Workshop on Applications of Semantic Web Technologies for E-Learning (SW-EL)*, 2004.

[67] Elizabeth Murphy and María A. Rodríguez-Manzanares. Teachers perspectives on motivation in high school distance education. *Journal Of Distance Education*, 23(3):1–24, 2009.

[68] Tom Murray. *Authoring Tools for Advanced Technologies Learning Environments: Toward cost-effective adaptive, interactive and intelligent educational software*, chapter

An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art, pages 493–546. Number 17. Kluwer Academic Publishers, Netherlands, 2003.

[69] Roberto Palhares. Anuário brasileiro estatístico de educação aberta e a distância. Technical report, Associação Brasileira de Educação a Distância, 2008.

[70] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. pages 333+. 2002.

[71] José Marques Pessoa, Hylson Vescovi Netto, and Crediné Silva de Menezes. Famcora: um framework para a construção de ambientes cooperativos inteligentes de apoio a aprendizagem na internet baseado em web services e agentes. In Sérgio Crespo C. S. Pinto, editor, *XIII Simpósio Brasileiro de Informática na Educação - SBIE 2002: Metodologias, tecnologias e aprendizagem dentro do cenário de informática na educação*. UNISINOS, 2002.

[72] Ig Ibert Bittencourt Santana Pinto. *Modelos e Ferramentas para a Construção de Sistemas Educacionais Adaptativos e Semânticos*. Tese de doutorado, Universidade Federal de Campina Grande, Campina Grande, 2009.

[73] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition, September 2005.

[74] PURE-SYSTEMS. pure-systems: pure::variants., 2010. Disponível em: <http://www.pure-systems.com/Community_Edition.55.0.html>. Acessado em 12/02/2010.

[75] Manuel Rodrigues, Paulo Novais, and Manuel Filipe Santos. Future challenges in intelligent tutoring systems : a framework. In *International Conference on Multimedia and Information and Communication Technologies in Education*, 2005.

[76] J. Jeffrey Rusk and Dragan Gasevic. Semantic web services-based reasoning in the design of software product lines. In *SPLC (2)*, pages 123–130, 2008.

[77] J. Jeffrey Rusk and Dragan Gasevic. Semantic web services-based reasoning in the design of software product lines. In *SPLC (2)*, pages 123–130, 2008.

[78] Kathleen Scalise and Bernard Gifford. Computer-based assessment in e-learning: A framework for constructing intermediate constraint questions and tasks for technology platforms. *The Journal of Technology, Learning, and Assessment*, 4(6), 2006.

[79] Stefan Schewe, Thomas Quak, Bettina Reinhardt, and Frank Puppe. Evaluation of a knowledge-based tutorial program in rheumatology - a part of mandatory course in internal medicine. In *ITS '96: Proceedings of the Third International Conference on Intelligent Tutoring Systems*, pages 531–539, London, UK, 1996. Springer-Verlag.

[80] SEI. Software engineering institute. Disponível em http://www.sei.cmu.edu/productlines/. Acessado em 27, Jan. de 2010.

[81] John Self. The defining characteristics of intelligent tutoring systems research: Itss care , precisely. *International Journal of Artificial Intelligence in Education*, 1999.

[82] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[83] Alan Silva, Evandro Costa, Ig Ibert Bittencourt, Patrick Brito, Olavo Holanda, and Diego Dermeval. Semantic web-based software product line for building inteligent tutoring systems. In *Second International Conference of Software, Services and Semantic Technologies*, pages 127–136, Setembro 2010.

[84] Marlos Silva, Endhe Elias, Ig Ibert Bittencourt, Heitor Barros, Douglas Véras, Evandro Costa, and Alan Pedro da Silva. Using a model driven approach for specifying educational ontologies. In *Proceedings of the 3rd Brazilian Workshop on Semantic Web and Education at SBIE'10*. SBC, 2010.

[85] Marlos Silva, Endhe Elias, Evandro Costa, Ig Ibert Bittencourt, Heitor Barros, Leandro Dias da Silva, Alan Pedro da Silva, and Douglas Véras. Combining methontology and a model driven ontology approach to build an educational ontology. *IEEE Multidisciplinary Engineering Education Magazine (MEEM)*, 2011.

[86] Goran Simic, Dragan Gasevic, and Vladan Devedzic. Semantic web and intelligent learning management systems. In *International Workshop on Applications of Semantic Web Technologies for E-Learning (SW-EL). In Intelligent Tutoring System, Maceió, Alagoas, Brazil*, 2004.

[87] D. Sleeman and J.S. Brown. *Intelligent Tutoring Systems*. Academic Press, Londres, 1982.

[88] Jennelle Irene Spurlock, Wendy Zhang, and Leetta Allen-Haynes. Can e-learning replace the traditional classroom? a case study at a private high school. In Newport, editor, *In The Proceedings of the Information Systems Education Conference*, volume 21, 2004.

[89] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Semantic web service discovery in the owl-s ide. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 109.2, Washington, DC, USA, 2006. IEEE Computer Society.

[90] Y. Vovides, S. Sanchez-Alonso, V. Mitropoulou, and G. Nickmans. The use of e-learning course management systems to support learning strategies and to improve self-regulated learning. *Educational Research Review*, 2(1):64–74, 2007.

[91] Eric Wang, Sung Ah Kim, and Yong Se Kim. A rule editing tool with support for non-programmers in an ontology-based intelligent tutoring system. In *Workshop on Semantic Web for e-Learning (SW-EL), 3rd Int'l. Semantic Web Conference (ISWC), Hiroshima, Japan*, 2004.

[92] Feng-Hsu Wang and Dai-Yan Chen. A knowledge integration framework for adaptive learning systems based on semantic web languages. In *ICALT '08: Proceedings of the 2008 Eighth IEEE International Conference on Advanced Learning Technologies*, pages 64–68, Washington, DC, USA, 2008. IEEE Computer Society.

[93] David M. Weiss and Chi. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley Professional; Har/Cdr edition, 1999.

[94] Beverly Park Woolf. *Building intelligent interactive tutors : student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann Publishers/Elsevier, Amsterdam ; Boston, 2009 2009.

[95] Beverly Park Woolf. *Building Intelligent Interactive Tutors Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann Publishers, 2009.

[96] Jennifer Yeo, Seng Chee Tan, and Yew-Jin Lee. A learning journey in problem-based learning. In *ICLS '06: Proceedings of the 7th international conference on Learning sciences*, pages 859–865. International Society of the Learning Sciences, 2006.

[97] Tao Zhang, Lei Deng, Jian Wu, Qiaoming Zhou, and Chunyan Ma. Some metrics for accessing quality of product line architecture. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02*, pages 500–503, Washington, DC, USA, 2008. IEEE Computer Society.