# REUTILIZAÇÃO EM SISTEMAS DE TUTORIA INTELIGENTE: EXPERIÊNCIA COM PADRÕES DE PROJETO, AGENTES INTELIGENTES E *FRAMEWORKS*

**Marco A. F. de Souza***

**Maria A. G. V. Ferreira****

## Resumo

Neste trabalho apresentam-se os resultados de uma experiência no estabelecimento de um *Framework* para Sistemas de Tutoria Inteligente, cuja finalidade é facilitar a implementação de novos tutores com a aplicação de técnicas de reutilização. *Frameworks* são estruturas de alto nível de abstração e, quando utilizados em projetos de novos sistemas, ganhos consideráveis de custo e tempo são obtidos, uma vez que eles descrevem soluções bem-sucedidas para problemas comuns encontrados em suas arquiteturas. Este artigo descreve o processo construtivo do *Framework*, justificando as decisões tomadas, apresentando a metodologia empregada e destacando os principais fatos observados durante o processo.

* Universidade Presbiteriana Mackenzie
E-mail: marco.souza@poli.usp.br
** Escola Politécnica da Universidade de São Paulo, PCS
E-mail: maria.alice.ferreira@poli.usp.br

## Abstract

In this work, the results of an experience in the development of a Framework for Intelligent Tutoring Systems are presented whose purpose is to facilitate the implementation of new tutors with the application of reuse techniques. Frameworks are high-level abstraction design structures and when used in the construction of new systems, considerable gains of cost and time are obtained since they describe successful solutions for common problems. This article describes the constructive process of the Framework, justifying the decisions taken, presenting the employed methodology and detaching the main facts observed during the process.

# 1 INTRODUÇÃO

Apesar do crescimento explosivo dos sistemas de comunicação e da disseminação da tecnologia da informática, que se uniram para apresentar aos educadores um novo panorama para a aprendizagem, a Informática Educativa encontra-se ainda em fase embrionária, muito aquém das suas reais possibilidades. Isso se deve a múltiplos fatores, que vão desde a diversidade e a multidisciplinaridade da área educacional até as dificuldades inerentes ao desenvolvimento de qualquer sistema computacional, ou seja, à complexidade que envolve a especificação e a implementação de sistemas de software. Dessa forma, o que acontece, na prática, é que os sistemas educacionais acabam sendo desenvolvidos sem grande planejamento, tornando a tarefa árdua e arriscada. Outro problema diz respeito à modelagem do conhecimento, que constitui outro tema de pesquisa, ainda em fase embrionária.

Em Educação, vem sendo dada ênfase especial à Educação a Distância, que seria melhor chamada de Educação Virtual Interativa (EVI),[1] pois, muitas vezes, não é necessário que exista "distância física" para que se recorra aos benefícios da introdução do computador no ambiente de aprendizagem. Dentro desse conceito novo, em que o aluno é o foco central do processo, os Sistemas de Tutoria Inteligente (STIs) podem ser de grande valia, quer seja nos ambientes de ensino tradicionais, quer seja nos ambientes de treinamento específico, existindo ou não distância física. A vantagem de uso desses sistemas pelos alunos é que eles permitem uma maior flexibilidade na aprendizagem, permitindo-lhes uma participação mais ativa do que a obtida com os sistemas CAI (*Computer-Aided Instruction* ou Instrução Auxiliada por Computador).

Reutilização de software é uma subárea da Engenharia de Software considerada promissora, mas ainda pouco explorada. Este trabalho descreve uma experiência em reutilização de software no campo dos Sistemas de Tutoria Inteligente, em que foram utilizados artefatos denominados Padrões de Projeto (*Design Patterns*). Artefato de software é o nome dado a qualquer parte de software que possa ser reutilizada no processo de desenvolvimento de novos programas, tais como rotinas, documentos, interfaces homem-máquina, especificações de requisitos e estruturas de análise e de projeto.[2] Padrões de Projeto são artefatos de software empregados na fase de Projeto do ciclo de desenvolvimento do software.[3] Por serem estruturas de alto nível de abstração em orientação a objetos, a sua reutilização permite reduções consideráveis de prazos e custos nos projetos de novos sistemas, pois representam soluções prontas para partes da arquitetura destes.

Pode-se considerar que o processo de reutilização pelo uso de Padrões de Projeto é composto por duas etapas: numa primeira etapa, descobrem-se e organizam-se

Padrões em um Catálogo para um dado domínio e, na etapa seguinte, constrói-se um software novo no domínio por meio da seleção e da adaptação de Padrões do Catálogo. Neste trabalho, descreve-se a experiência do uso de Padrões de Projeto na elaboração de um *Framework* para o domínio de Sistemas de Tutoria Inteligente. Posteriormente, o *Framework* produzido pode ser utilizado na construção de tutores para diferentes áreas de ensino-aprendizado: fundamentos de Matemática ou Geometria, ensino de linguagens de programação ou de metodologias de construção de sistemas de software.

Este artigo está dividido nas seguintes seções: na Seção 2 comenta-se o processo de desenvolvimento de sistemas de software empregando a reutilização; na Seção 3 discute-se a utilização do paradigma de Orientação a Objetos, Padrões de Projeto, *Frameworks* e STIs; na Seção 4 discute-se o processo de estabelecimento de um *Framework* para um sistema genérico de tutoria inteligente, e na Seção 5 são apresentadas algumas conclusões sobre o trabalho desenvolvido.

## 2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE E REUTILIZAÇÃO

A busca por metodologias de desenvolvimento de sistemas computacionais, que culminou com o nascimento da Engenharia de Software, data da década de 70. Para Pressman,[4] "sob o aspecto técnico, a Engenharia de Software se inicia com uma série de tarefas de modelagem que levam a uma especificação completa de requisitos e a uma representação compreensível do projeto do software a ser construído". A representação do sistema é feita por meio de modelos baseados em algum paradigma computacional. Existem vários paradigmas, porém fortes tendências indicam que a Orientação a Objetos será o paradigma dominante no futuro.

Um tópico essencial, associado ao processo de desenvolvimento de software, é a reutilização dos produtos gerados ao longo desse processo. Para Freeman,[5] *apud* Pressman,[4] "reutilização é simplesmente qualquer procedimento que produz (ou ajuda a produzir) um sistema, reutilizando-se alguma coisa de um esforço de desenvolvimento anterior".

Apesar da simplicidade aparente dessa idéia, até hoje a reutilização de software ainda é pouco praticada e conhecida, principalmente na área educacional. Entretanto, nos últimos anos, muitos esforços vêm se concentrando na reutilização de software, perseguindo não só os aspectos técnicos, mas também os gerenciais e mercadológicos.

Pesquisadores que se preocupam com o aspecto gerencial sugerem um modelo de desenvolvimento de sistemas baseado em dois estágios: o ciclo de vida para desenvolver os artefatos reutilizáveis e o ciclo de vida para desenvolver novos sistemas com os

artefatos reutilizáveis produzidos no primeiro estágio,[6,7] A Figura 1, adaptada de Hedenäng et al.,[8] reapresenta esse modelo. No ciclo de vida para desenvolver os artefatos reutilizáveis, os estágios de desenvolvimento do software partem da etapa de Análise de Domínio visando a uma família de aplicações e aplicações futuras. Já no ciclo de vida, para desenvolver com artefatos reutilizáveis, são utilizados os artefatos reutilizáveis disponíveis no repositório, e o produto final desse ciclo é uma nova aplicação, que possuirá a arquitetura definida pela combinação dos artefatos do repositório.

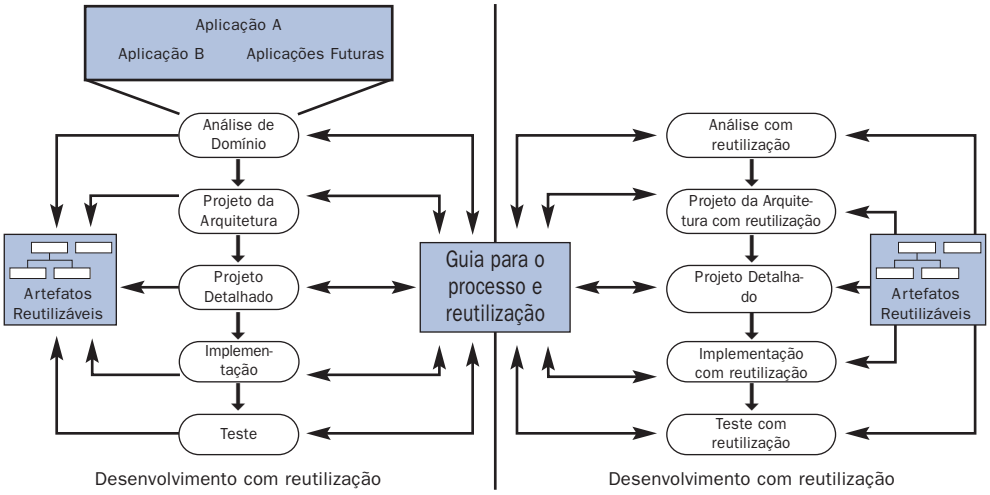


**Figura 1** Ciclo de desenvolvimento do software para e com reutilização

Nessa figura, apresentam-se as principais etapas utilizadas nos processos de desenvolvimento de software em cada um dos estágios. Essas etapas, representadas pelos retângulos de cantos arredondados, são similares em ambos os estágios, porém diferem quanto às técnicas empregadas em cada estágio. Em ambos os ciclos é primordial a existência de um guia que oriente o processo de reutilização (colocado no quadrado central) e o uso de um repositório de artefatos reutilizáveis (representado pelos retângulos à direita e à esquerda do desenho), em que são armazenados os produtos gerados nas várias etapas do ciclo de desenvolvimento de artefatos. Cabe ressaltar ainda que, apesar de não explícitas, existem na figura várias realimentações entre as várias etapas do ciclo, permitindo-se o retorno de uma dada etapa a qualquer uma das etapas anteriores, como em outros ciclos de vida mais conhecidos.

Este trabalho está centrado no estágio de Desenvolvimento para reutilização. Sendo *Frameworks* artefatos da fase de Projeto, as etapas percorridas se detêm no retângulo de Projeto Detalhado, e os elementos produzidos até esta etapa são armazenados no repositório.

# 3 ORIENTAÇÃO A OBJETOS, PADRÕES DE PROJETO, FRAMEWORKS E STIs

Visando a diminuir as dificuldades existentes na criação e reutilização de STIs, partiu-se para um trabalho de criação de um *Framework* para esses sistemas, que pudesse ser empregado no desenvolvimento de novos tutores.

Um *Framework* é uma coleção de classes que provê um conjunto de serviços, geralmente, para um domínio particular. Vários *Frameworks* são incompletos ou abstratos e são completados com a incorporação de componentes que permitirão sua adaptação a um problema específico. Um *Framework* pode ser reutilizado para criar várias aplicações similares.

Apesar de *Frameworks* e bibliotecas de classes serem similares em princípio, existem diferenças que os separam. Um *Framework* possui em geral o controle do fluxo de execução de seus objetos, define como será feita a interação entre os objetos, bem como realiza uma inicialização padrão, características não encontradas geralmente nas bibliotecas de classes, que exigem das aplicações clientes grande parte da responsabilidade de controlar seus objetos. Um *Framework* apresenta assim uma certa funcionalidade, caracterizando uma estrutura de nível funcional mais alto que as classes.

Padrões de Projeto são estruturas que se repetem em diferentes projetos de sistemas de software – similares ou não – e que podem ser reutilizadas posteriormente em novos projetos. O objetivo de um Padrão é identificar essas partes e catalogá-las de forma sistemática. Assim, na necessidade de resolver um novo problema, o projetista pode utilizar um desses Padrões como solução, sem ter de redescobri-la novamente.

Os Padrões também são constituídos por classes e objetos – conforme definição do paradigma de Orientação a Objetos – e indicam como esses elementos estão organizados para atender ao propósito pretendido pelo projetista.[3] São determinados por meio de observações e de um processo de tentativa e erro, observando-se os blocos construtivos básicos de várias aplicações e seus respectivos relacionamentos.[9] Novos Padrões podem ser descritos por Padrões já existentes.

Padrões podem ser utilizados para descrever *Frameworks*. Um *Framework* pode ser construído baseando-se em vários padrões, mas o inverso não é verdadeiro. Um padrão é sempre intencionado a ser mais genérico que um *Framework*.

A escolha de *Frameworks* para o desenvolvimento de sistemas educacionais apresenta inúmeras vantagens. As linguagens de programação mais modernas, providas de ambientes de desenvolvimento amigáveis, estão voltadas para a utilização desses artefatos, como C++ Builder, DELPHI, Visual C++ ou JAVA (que vem sendo usada

intensamente na geração de programas para a Web, muitos dos quais destinados à Educação a Distância). Finalmente, as linguagens orientadas a objetos oferecem recursos que facilitam a reutilização de código, tais como composição e herança.

A escolha de *Frameworks* e Padrões de Projeto como técnica a ser utilizada na descrição da arquitetura genérica de um STI deve-se primeiramente ao fato de que a reutilização de artefatos de níveis mais altos de abstração permite maiores ganhos do que a reutilização de componentes mais básicos, como rotinas. Segundo Jacobson, Griss e Jonsonn,[10] escrever código custa entre 10% e 20% do custo total, tornando os ganhos com esse tipo de reutilização relativamente baixos. Como Padrões e *Frameworks* são constituídos por um conjunto de classes e relacionamentos já estabelecidos, resolvendo problemas específicos de um determinado tipo de projeto e que podem ser adaptados de forma flexível a novos sistemas, trabalha-se em um nível mais alto de abstração, reduzindo-se assim o tempo de desenvolvimento e os custos envolvidos em percentagens mais significativas.

Em segundo lugar, a utilização desses artefatos obriga um processo de análise de domínio, no qual a arquitetura é analisada de acordo com as possíveis variações que deverá apresentar nos contextos em que for utilizada. Portanto, como uma conseqüência natural, há uma descoberta de componentes potencialmente reutilizáveis da arquitetura. Estes, por um processo de composição e de adaptação, podem servir como base para a criação de uma biblioteca de componentes reutilizáveis para algum sistema a ser construído com essa arquitetura. Assim, em sistemas complexos como Sistemas de Tutoria Inteligente, a identificação e a criação de componentes reutilizáveis permitirão que estes sejam aproveitados em toda a extensão da arquitetura proposta.

Em terceiro lugar, o uso de Padrões de Projeto na construção de *Frameworks* permite utilizar soluções de projeto já testadas e aprovadas por projetistas experientes (são catalogadas), implicando que por trás desses padrões exista uma confiabilidade e qualidade agregadas em sua solução, que será transportada pela aplicação do Padrão. Além disso, a utilização de Padrões de Projeto implica também o nivelamento de informações de projeto entre os diversos membros de uma equipe de desenvolvimento, estabelecendo vocabulário e entendimento comuns de técnicas de projeto em um espaço de tempo reduzido.[11]

Por fim, os Padrões de Projeto podem ser utilizados no processo de treinamento dos engenheiros de software envolvidos no projeto, possibilitando a rápida adequação de novos engenheiros com a arquitetura do sistema e com decisões de projeto a serem tomadas no transcurso deste. Além disso, como cada Padrão se refere a um problema específico, há o benefício adicional de atenuar a carga cognitiva do aprendizado.[12]

# 4 ESTABELECENDO UM *FRAMEWORK* PARA STI

O processo de estabelecimento de um *Framework* é constituído por uma série de etapas que são executadas de forma consecutiva, de acordo com o ciclo de vida apresentado na Figura 1. São elas:

- análise de domínio, que permite identificar as classes e os relacionamentos comuns a diversos STIs construídos, gerando, por conseqüência, uma arquitetura genérica;
- projeto da arquitetura do software, no qual são estabelecidos os relacionamentos entre as classes escolhidas;
- projeto detalhado das classes (estabelecimento do *Framework*).

Esses três passos serão descritos a seguir.

## 4.1 Análise de domínio

A arquitetura genérica adotada para o sistema STI tem como inspiração o processo de ensino empregado por humanos. Neste, estão envolvidos os seguintes tipos de conhecimento:

- O que ensinar – corresponde ao domínio do assunto a ser ensinado.
- Como ensinar – diz respeito às técnicas de ensino empregadas pelo tutor, que, dependendo da avaliação corrente do Estudante, escolherá partes apropriadas do domínio e maneiras de ensinar.
- A quem ensinar – diz respeito ao conhecimento do Estudante, que servirá como maneira de avaliá-lo e de guiar o tutor nas suas decisões pedagógicas.

Essa separação do conhecimento reflete-se na arquitetura, ilustrada na Figura 2, de um sistema STI computadorizado em quatro componentes básicos, aqui descritos de forma sucinta:[13]

- Componente de Domínio: armazena conhecimento procedimental e declarativo sobre o domínio que o *estudante* deseja estudar. Normalmente esse conhecimento é especificado por um especialista no domínio desejado;
- Componente de Tutor: contém um mecanismo de inferência que, com base em informações sobre o entendimento corrente do estudante, toma decisões sobre como progredir no processo de tutoria;

- Componente do estudante: contém informações que caracterizam o entendimento do estudante em relação ao domínio que está sendo ensinado, aliado a um mecanismo de inferência que de alguma forma compara o conhecimento do estudante com o conhecimento de domínio, diagnosticando o estado de entendimento atual do estudante. Essas informações são então utilizadas para a tomada de decisões pedagógicas pelo Componente de Tutor;
- Componente de Interface: permite a interação do estudante com o tutor.

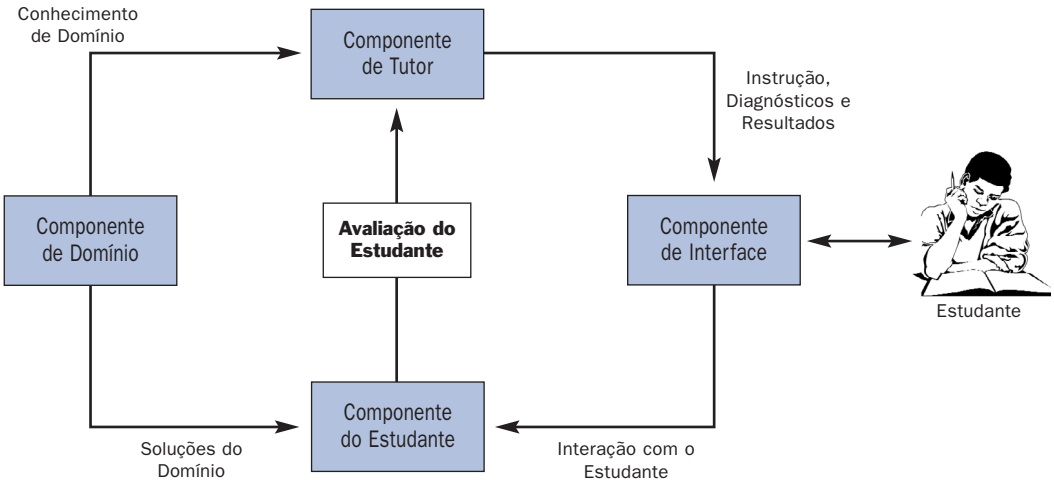


**Figura 2** Componentes básicos de um STI

A Análise de Domínio fornece uma metodologia para identificar artefatos reutilizáveis dentro de um domínio específico, destacando e classificando, em diversos projetos analisados, as partes que são comuns e, portanto, potenciais candidatas a ser reutilizadas.[14] Assim, as informações resultantes da Análise de Domínio podem ser utilizadas como base para a identificação de Padrões a serem utilizados na construção de *Frameworks*.[15]

Uma etapa necessária à Análise de Domínio foi a eliciação dos Requisitos Funcionais do STI, baseado na arquitetura genérica dele – neste processo, expressos em termos de Casos de Uso. Foram definidos cinco Casos de Uso operados diretamente pelo *estudante* – o ator principal do sistema. Esses Casos de Uso, que estão mostrados na Figura 3, à esquerda, foram, então, detalhados um a um, o que permitiu a descoberta e a organização de funções que não apareciam na arquitetura genérica inicial de alto nível.

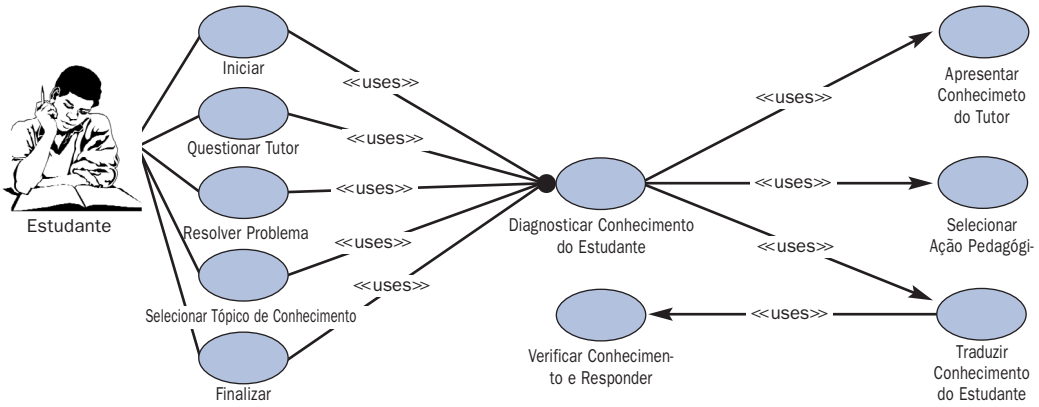


**Figura 3** Diagrama de Casos de Uso do Tutor Inteligente Genérico Padrão

## 4.3 Projeto da arquitetura de software

Cada Caso de Uso apresentado na Seção 4.2 foi refinado e descrito de forma detalhada textualmente. Desses textos foram extraídos classes, relacionamentos e métodos que permitiram a concretização desses Casos de Uso.[16] Essa primeira arquitetura foi então refinada com o auxílio de Padrões de Projeto, permitindo, assim, a identificação de elementos reutilizáveis.

Para a representação dos tipos de conhecimento do tutor e para a realização de inferências, optou-se por utilizar como inspiração a arquitetura Quadro-Negro, com um Sistema Baseado em Regras, por permitir um maior desacoplamento entre as bases de conhecimento e os mecanismos de inferência, que, doravante, serão denominados de agentes inteligentes baseados em regras.

O *Framework* proposto para STIs foi organizado em unidades separadas e está descrito na Figura 4, na qual uma seta tracejada representa uma dependência direta entre as unidades. Essas unidades são:

- Arquitetura de Tutor Inteligente: representa uma arquitetura genérica para um STI, inspirada em Sistemas Baseados em Regras;
- Agente Inteligente Baseado em Regras: descreve os agentes do sistema que participarão do *Framework* Arquitetura de Tutor Inteligente;
- Organização do Domínio: especifica uma organização para os elementos de domínio de um STI;
- Entendimento e Avaliação do Estudante: define uma organização para representar o estudante;
- Planos de Ensino: especifica a organização de regras de tutoria;
- Padrão Interface do Tutor: especifica uma arquitetura para a interface.

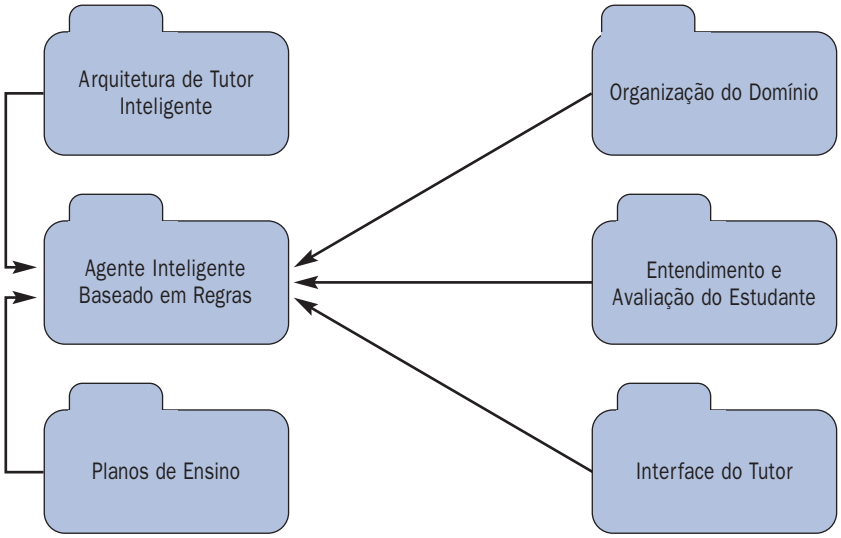


**Figura 4** *Framework* para Sistemas de Tutoria Inteligente

## 4.4 Exemplo – a Arquitetura de Tutor Inteligente

As unidades apresentadas na Figura 4 foram então detalhadas. Cada uma delas foi documentada de acordo com um formato consistente e de acordo com a proposta de Gamma et al.[3] Devido ao espaço restrito deste trabalho, será exemplificado somente o núcleo do *Framework*, Arquitetura de Tutor Inteligente, de forma resumida. Uma descrição completa pode ser encontrada no trabalho de Souza.[16]

O *Framework* Arquitetura de Tutor Inteligente representa uma arquitetura genérica, baseada em regras para um STI. Seu objetivo é prover uma arquitetura básica do núcleo de um tutor inteligente de forma que seus componentes possam ser entendidos e reutilizados com certa facilidade. Foi adotada a arquitetura de um sistema Quadro-Negro como base para a arquitetura do tutor.[17,18]

Aqui, as diversas fontes de conhecimento do tutor lêem e escrevem suas conclusões no quadro-negro, interagindo indiretamente. O quadro-negro poderia ser dividido em áreas, que seriam as etapas da construção da solução, em que cada etapa é concluída pela contribuição das fontes de conhecimento.

A estrutura da Arquitetura do Tutor Inteligente será separada em duas partes, para melhor compreensão. Na Figura 5 é exibida a organização de classes para realizar inferências no Quadro-Negro, enquanto na Figura 6 é exibida a hierarquia de classes dos agentes do *Framework*, que indica como os agentes dessa arquitetura são classificados no *Framework*.

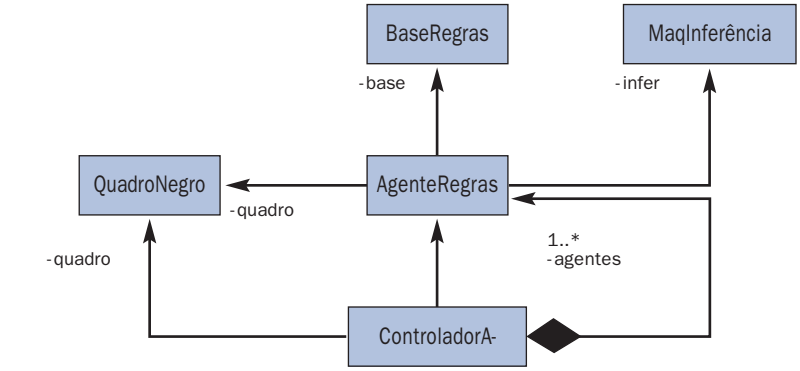


**Figura 5** Organização de classes para o Quadro-Negro

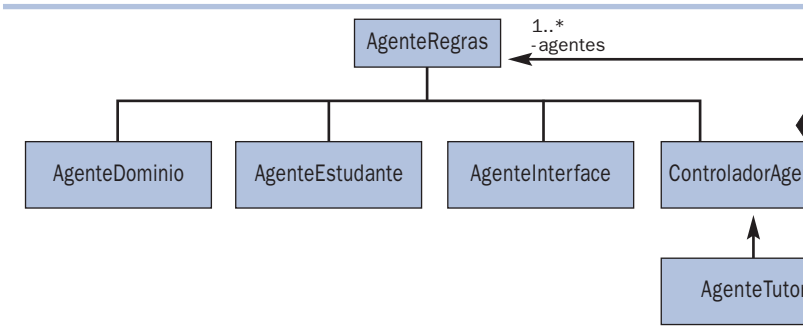


**Figura 6** Hierarquia de classes para os agentes do *Framework*

As classes do Quadro-Negro são, conforme mostrado na Figura 5:

- AgenteRegras: representa a interface de um agente inteligente baseado em regras. Seu objetivo é executar inferências sobre os dados no quadro-negro, utilizando-se de sua base de conhecimento que contém suas regras (classe BaseRegras) e realizando inferências com sua máquina de inferência (classe MaqInferencia).
- ControladorAgentes: questiona os agentes do sistema para verificar se eles podem contribuir na solução de um problema, controlando a execução das regras, quando ativadas. O controlador possui regras específicas para executar sua tarefa, daí ser herdeiro da classe AgenteRegra.
- QuadroNegro: armazena os fatos gerados durante o processo de solução de um determinado problema. Ele está associado aos agentes que o manipulam e pode hierarquizar o armazenamento dos fatos que contém, de modo que segmente partes da solução a ser determinada.

- BaseRegras: contém e gerencia as regras de cada agente. É utilizada pelo mecanismo de inferência de cada agente (classe MaqInferencia) para verificar quais de suas regras poderão contribuir com a solução no QuadroNegro.
- MaqInferencia: responsável pela inferência de novos fatos ou pela execução ("disparo") de regras de acordo com fatos existentes na base de regras de cada agente (classe BaseRegras). Cada agente possui sua própria máquina de inferência.

As classes restantes para a Arquitetura do Tutor Inteligente estão apresentadas na Fgura 6:

- AgenteDomínio: é herdeiro de AgenteRegras e tem como objetivo servir como referência nas soluções de problemas propostas pelo AgenteTutor.
- AgenteEstudante: é herdeiro de AgenteRegras e tem como objetivo a inferência do estado do estudante durante uma sessão de tutoria. Essa inferência implica uma tarefa de diagnóstico sobre os dados presentes no QuadroNegro.
- AgenteInterface: é herdeiro de AgenteRegras e é especializado em traduzir os comandos provindos da interface do sistema em uma representação de conhecimento inteligível ao quadro-negro e aos agentes. Possui um conjunto de regras especializadas em realizar a tarefa oposta, também, traduzindo o conhecimento interno armazenado no quadro-negro em comandos de interface, quando pertinente.
- AgenteTutor: é herdeiro de ControladorAgentes e representa o coordenador do sistema. O AgenteTutor possui regras para controlar a execução dos outros agentes, realizando uma tarefa de planejamento cujo objetivo é conseguir levar o estudante a um determinado estágio de conhecimento.

# 5 CONCLUSÕES

O *Framework* apresentado reflete uma visão de Engenharia de Software de apenas alguns aspectos de sistemas STIs e, mesmo assim, gerou uma quantidade substancial de informação. No entanto, com esses aspectos foi possível explicitar melhor a estrutura de um STI e as implicações relacionadas com seu projeto. Cabe ainda ressaltar que o *Framework* desenvolvido apresenta a característica da expansibilidade, podendo-se acrescentar novas características, não diretamente relacionadas à Engenharia de Software, que tratem de temas como Psicologia Cognitiva e Educação, por exemplo.

Desenvolvimentos a mais logo prazo prevêem a incorporação de características que permitiriam a adaptação a ambientes distribuídos, bem como a melhoria do agente inteligente, de modo que ele possa ser adaptado a outros paradigmas de Inteligência Artificial.

## 6 AGRADECIMENTOS

Ao CNPq, que suportou parcialmente esta pesquisa.

## REFERÊNCIAS BIBLIOGRÁFICAS

1. TORI, R.; FERREIRA, M. A. G. V. Educação a distância em cursos de informática. In: XV Congresso da Sociedade Brasileira de Computação – *VII Workshop sobre Educação em Informática*. Rio de Janeiro, 19 a 21 de julho de 1999. *Anais.* p. 581-590.
2. KRUEGER, C. W. Software reuse. *ACM Computing Surveys*. v. 24, n. 2, jun. 1992.
3. GAMMA, E. et al. *Design patterns* – elements of reusable object-oriented software. Addison-Wesley Publishing Company, 1995.
4. PRESSMAN, R. S. *Software engineering* – a practitioner's approach. 4. ed. New York: McGraw-Hill, 1997.
5. FREEMAN, P. A Perspective on reusability. In: FREEMAN, P. (Ed.) *Software Reusability*. *IEEE Computer Society Press*, 1987. p. 2-3.
6. MILI, H.; MILI, F.; MILI, A. Reusing software: issues and research directions. *IEEE Transactions on Software Engineering*, v. 21, n. 6, jun. 1995.
7. McCLURE, C. *Software Reuse Techniques – Adding reuse to the systems development process*. Prentice-Hall PTR, 1997.
8. HEDENÄNG, R. et al. Development for reuse. In: KARLSSON, E. (Ed.). *Software Reuse* – A Holistic Approach. John Wiley & Sons, 1995, cap. 9, p. 287-342.
9. COAD, P. Object-oriented patterns. *Communications of the ACM*. v. 35, n. 9, p. 152-159, set. 1992.
10. JACOBSON, I.; GRISS, M.; JONSONN, P. Making the reuse business work. *Computer*, v. 29, n. 10, p. 36-42, oct. 1997.

11. SOUZA, M. A. F. de.; FERREIRA, M. A. G. V. Aplicação de Heurísticas e Padrões de Projeto nas Metodologias Orientadas a Objetos. In: *WORKCOMP'99*, Instituto Tecnológico de Aeronáutica, 5 a 6 de outubro de 1999.

12. SOUZA, M. A. F. de, FERREIRA, M. A. G. V. Educando o Engenheiro de Sotware através de Heurísticas e Padrões de Projeto. In: *VII Congresso Iberoamericano de Educación Superior en Computación, CIESC'99*. Assunção, Paraguai, 29 de agosto a 3 de setembro de 1999.

13. WENGER, E. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. Morgan Kaufmann, 1987.

14. PRIETO-DÍAZ, R., ARANGO, G. Domain Analysis Concepts and Research Directions. In: Prieto-Díaz, R., Arango, G. (Ed.). *Domain Analysis and Software Systems Modelling*. IEEE Computer Society Press, 1991, p. 9-32.

15. FRASER, S.; LEISHMAN, D.; McLELLAN, R. Patterns, Teams and Domain Engineering. In: *ACM SIGSOFT Symposium on Software Reusability – SSR'95*. p.222-224, 1995.

16. SOUZA, M. A. F de. *Arquiteturas reutilizáveis para a criação de sistemas de tutorização inteligentes*. (Dissertação, Mestrado). Escola Politécnica, Universidade de São Paulo. São Paulo, 2000.

17. HAYES-ROTH, B. A Blackboard Architecture for Control. *Artificial Intelligence*, n. 26, p. 251-321, 1985.

18. LUGER, G. F.; STUBBLEFIELD, W. A. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Benjamin/Cummings, 1993.