Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação

Carlos Miguel Tobar	João Luís Garcia Rosa
Juan Manuel Adán Coello	Ricardo Pannain

Instituto de Informática - PUC-Campinas
{tobar,joaol,juan,pannain}@ii.puc-campinas.br

Resumo

Um dos problemas encontrados nos cursos de Computação e Informática é o aprendizado de programação nas disciplinas introdutórias. Com o objetivo de tentar resolver este problema, propõe-se um ambiente colaborativo para aprendizado de programação, seguindo o paradigma procedimental, que deve permitir o envolvimento de estudantes, professores e sistemas inteligentes, oferecendo meios para geração e discussão de idéias, resolução de problemas, acesso e localização de informação *on-line* útil, e motivação à participação dos estudantes.

Palavras-Chave: Informática na Educação Superior, Aprendizagem Colaborativa Apoiada por Computadores, Aprendizado de Programação, Arquiteturas Distribuídas para *Software* Educativo

1. Introdução

Cursos de graduação na área de Computação e Informática, com denominações recomendadas para Bacharelado em Ciência da Computação, Engenharia de Computação, Bacharelado em Sistemas de Informação e Licenciatura em Computação (MEC/SESu, 1997), tradicionalmente adotam o modelo de ensino em que a programação é priorizada nas disciplinas introdutórias.

O modelo que prioriza a programação não é único. Segundo a *Joint Task Force on Computing Curricula* (*Computer Society Connection*, 2001), existem outros modelos que podem ser usados para a definição de currículos. Além disso, o modelo de programação atualmente pode ser implementado através de três diferentes estratégias, de acordo com o paradigma de programação adotado: procedimental, orientado a objetos ou funcional.

Proulx (2000) observou que um grave e freqüente problema vivenciado por estudantes é a primeira experiência no aprendizado de programação, que se transforma em uma grande barreira para vários estudantes. Os motivos para essa experiência frustrante são vários: a preocupação excessiva com detalhes de sintaxe da linguagem sendo usada; a falta de uma visão daquilo que se quer solucionar, de idealizar soluções adequadas, de mapear essas soluções em passos seqüenciais e de abstrair o funcionamento dos mecanismos escolhidos; o estabelecimento de um raciocínio lógico visando a resolução de problemas, com base em um modelo incremental, em relação à complexidade e à estratégia de refinamentos sucessivos.

As dificuldades encontradas podem ser percebidas através do alto grau de repetência e das dificuldades demonstradas nas disciplinas diretamente dependentes das habilidades de programar, de dominar o raciocínio lógico e de resolver problemas. Vários são os estudantes que completam seus cursos sem essas habilidades. Em parte, isso é decorrência da dificuldade encontrada pelos professores para acompanharem efetivamente as atividades laboratoriais de programação, dado o grande número de

estudantes geralmente sob sua supervisão. O problema se agrava, muitas vezes, quando essas atividades são desenvolvidas em grupo. Isto parece paradoxal, pois o trabalho em grupo deveria propiciar as condições para o exercício do aprendizado colaborativo e que tem sido correlacionado com uma série de resultados positivos, entre eles, maior aprendizado, maior produtividade, transferência de conhecimento e desenvolvimento do senso de competência (Suthers, 1998). Entretanto, não basta oferecer mecanismos de comunicação entre os estudantes para que haja colaboração. As ferramentas automatizadas devem ser adequadamente projetadas para esse fim. Quando isso é feito, tem sido constatado que os estudantes conseguem alcançar melhorias no desempenho, no pensamento crítico e no comportamento cooperativo (Gokhale, 1995).

O projeto Suporte Tecnológico para o Aprendizado Colaborativo de Desenvolvimento de *Software* visa a definição, o projeto (*design*) e a implementação de um ambiente de aprendizado colaborativo, mediado por computador, voltado para a área de desenvolvimento de *software*, em um primeiro momento, seguindo o paradigma procedimental. O público alvo deste ambiente é dos estudantes de cursos de Computação e Informática.

Este ambiente deve transcender à proposta do *Computer Science Education Research Group* (Rasala *et al*., 2000), que se propõe a desenvolver um ambiente na *Web*, constituído por diversos componentes, tais como, *toolkits*, projetos, *patterns*, exercícios práticos e documentação, para linguagens orientadas a objetos (C++ e Java). O ambiente objeto deste trabalho deve, além disso, permitir o envolvimento cooperativo de diferentes atores: estudantes, professores e sistemas inteligentes, dentro e fora de sala de aula, oferecendo meios para: (1) geração e discussão de idéias, (2) resolução de problemas, (3) acesso e localização de informação *on-line* útil, e (4) motivação à participação dos estudantes.

A arquitetura proposta para tal ambiente pode ser caracterizada a partir de três dimensões: 1) as tecnologias que oferecerão os meios acima listados ao aprendizado colaborativo; 2) uma hierarquia de níveis de abstração, para servir de referência à implementação do ambiente que apresenta natureza distribuída; e 3) os módulos funcionais que comporão o ambiente.

A seguir, na seção 2, apresenta-se um cenário de uso do ambiente e os respectivos pressupostos pedagógicos. Na seção 3, é apresentada a arquitetura do ambiente, a partir das três dimensões que o caracterizam. Concluindo o artigo, na seção 4, são feitas algumas considerações finais, indicando o estágio em que se encontra o esforço para criação deste ambiente.

2. Um cenário de uso do ambiente

O ambiente pretendido deve oferecer um conjunto de serviços e mecanismos que possam ser combinados pelo instrutor para suportar diversas metodologias de ensino. Neste momento, os cenários de uso do ambiente e, conseqüentemente, as ferramentas a implementar partem dos seguintes pressupostos pedagógicos:

- • Os estudantes aprendem a partir das suas controvérsias quando discutem pontos de vista diferentes, apresentam alternativas e, além disso,pedem e dão explicações. Este ponto de vista é suportado pela teoria do conflito sócio-cognitivo, conforme aplicado no sistema COOLER (Constantino-Gonzalez e Suthers, 2000), voltado ao aprendizado colaborativo de modelagem entidade-relacionamento para bancos de dados.
- • Aprender a programar envolve aprender a construir mecanismos e explicações. Esta postura, segundo Soloway (1986), considera que um programa

tem duas audiências, o computador e o leitor humano. As instruções de um programa transformam o computador em um mecanismo que indica *como* resolver o problema. Por sua vez, o leitor humano, o programador, precisa de uma explicação de *porquê* o programa resolve o problema. Na realidade, a necessidade de construir mecanismos e explicações transcende a área de programação, sendo aplicável a inúmeras situações que envolvem a solução de problemas.

- • O teste de programas ajuda a aumentar a funcionalidade de programas, além disso, a leitura e revisão de código pelos pares aumenta a qualidade de programas e do estilo dos programadores. Estes pressupostos foram usados no sistema Praktomat (Zeller, 2000) com bons resultados relatados. Tratam-se de práticas industriais, sendo a revisão ainda um dos mais efetivos meios de se obter código compreensível e de fácil manutenção.

O uso do ambiente pode ser caracterizado, primeiramente, por um cenário com três etapas, não necessariamente seqüenciais: 1) aquisição de informação e conhecimento básico sobre um determinado tópico pelo estudante; 2) consolidação desse conhecimento pelo estudante, através da solução cooperativa de problemas; e 3) realimentação e aprendizado pelo sistema, através da interação com o estudante.

O instrutor poderá intervir em qualquer etapa, a partir de suas próprias observações, na medida em que poderá ter acesso às áreas de trabalho de estudantes e grupos, ou por sugestão do sistema. Neste último caso, será necessário estabelecer políticas e mecanismos para determinar quais seriam os momentos oportunos em que isso deveria acontecer.

2.1. Aquisição de Informação e Conhecimento Básico

O estudante que deseja adquirir conhecimento sobre determinado tópico de programação, apresenta sua intenção ao sistema. Este oferece mecanismos para extração, tratamento semântico, filtragem, validação, transformação, classificação e mineração de dados.

O ambiente apresenta ao estudante opções de materiais que tratam do tópico. Essas opções ou objetos educacionais podem adquirir a forma de textos introdutórios, explicações, exercícios ou exemplos. Os objetos educacionais são objetos complexos, que permitem diferentes perspectivas para apresentação e podem ser constituídos de informações sobre desempenho, ocupação de espaço em memória, além de anotações e documentação, gerada através de diferentes mídias (*e-mail*, documento, etc.).

O sistema deve oferece a possibilidade de relacionar objetos educacionais para as mais variadas finalidades, por exemplo, aqueles objetos que em conjunto podem satisfazer algum requisito do tipo "ordenação sobre vetores, na ordem de log n ou melhor", ou que podem ser considerados como componentes de um algoritmo mais complexo, ou que foram produzidos pelos estudantes de determinada disciplina em determinada data, ou que constituem recomendações para solução do problema sendo enfocado.

Além de objetos educacionais, o estudante tem acesso a referências para livros, artigos ou materiais na *Web*, e contato com pares, instrutor ou especialistas, através de canais de comunicação.

2.2. Solução Cooperativa de Problemas

Considerando que tenham adquirido o conhecimento suficiente sobre um determinado tema, os estudantes passam à etapa de consolidação e aprofundamento desse conhecimento, através da solução de problemas propostos pelo ambiente. Inicialmente,

os estudantes produzem soluções e explicações individuais, visando preparar-se para uma colaboração efetiva com seus pares. Os programas individualmente produzidos são submetidos ao sistema que os compila e testa. Feito isso, os estudantes passam a um processo de colaboração visando à produção de soluções "finais" individuais ou coletivas.

Produzida uma solução "final", esta é apresentada ao sistema, passando-se à etapa de realimentação do sistema, a ser discutida adiante. Em síntese, a solução de problemas de programação poderia ser dividida em quatro passos: (1) Proposição do problema, (2) Elaboração individual de um programa e de uma explicação, (3) Submissão pelo estudante e teste pelo sistema do programa, (4) Colaboração entre os estudantes, mediada pelo sistema.

No passo 3, após a submissão das soluções individuais, o sistema procurará em primeiro lugar, verificar possíveis plágios, que poderiam comprometer todo o processo de colaboração, na medida em que alguns dos estudantes não teriam de fato se preparado para assumir uma postura ativa durante o etapa de colaboração. Ao copiarem um programa, ou parte dele, os estudantes muito provavelmente não refletiram o suficiente sobre a solução do problema proposto. Descartado o plágio, o sistema compila e executa o programa, submetendo-o a alguns testes escolhidos dentre um conjunto previamente definido pelo instrutor.

Na etapa de colaboração (passo 4) poderiam ser usadas estratégias básicas, isoladamente ou combinadas:

- As soluções propostas são comparadas e o sistema coloca em contato estudantes que apresentam soluções com diferenças substanciais. As diferenças devem ser discutidas e deve-se buscar uma solução de consenso. Este passo pode ser repetido várias vezes.
- Os estudantes são reunidos em grupos para desenvolver uma solução conjunta para o problema. Durante o desenvolvimento da solução, o sistema procura estimular a participação de elementos cujas soluções individuais estejam significativamente diferentes da solução da maioria do grupo.
- Cada programa submetido é comentado por um ou mais estudantes. A partir dos comentários, o autor do programa poderá então revisar o seu código, a fim de melhorá-lo. Este passo pode ser repetido várias vezes. Esta estratégia pode ser combinada com as anteriores ou usada de forma independente. Neste caso, pode ser interessante propor problemas diferentes aos estudantes no passo 1.

2.3. Realimentação e Aprendizado pelo Sistema

Feita a submissão final de uma solução, o sistema procura numa memória, ou base de soluções (geradas e comentadas por outros estudantes e pelos instrutores), programas semelhantes ao submetido. Os comentários adicionados às soluções recuperadas podem ter várias dimensões, por exemplo, fazendo menção à qualidade da solução e indicando possíveis melhorias. Os estudantes podem adicionar comentários às soluções da base que lhe forem apresentadas. Esses comentários podem ou não ser mantidos após a apreciação do instrutor. Opcionalmente, poderia também ser interessante apresentar soluções substancialmente diferentes das submetidas (melhores ou não).

A utilização da memória do sistema poderia naturalmente ser também usada durante a etapa de colaboração. Neste caso, o sistema adotaria uma postura ativa nesse processo, portando-se, em certo sentido, como mais um elemento no processo de solução.

3. Uma Arquitetura Multidimensional e suas Diretrizes

Todos os esforços a serem realizados como decorrência da proposta da arquitetura, visando a definição, projeto e implementação de um ambiente colaborativo para aprendizado, deverão observar o seguinte conjunto de diretrizes:

- • Colaboração - a concepção de espaços de aprendizagem mediados por computador, dentro de uma orientação colaborativa decorrente da natureza social do aprendizado, constitui uma proposta alternativa e contemporânea da prática pedagógica articulada à construção de pedagogias transformativas (Lite, 2000).
- • Personalização - segundo Pednault (2000), a extensão em que um produto ou serviço interativo personalizado pode se adaptar a usuários individuais depende da informação capturada pelo sistema e de como essa informação é utilizada; a verdadeira personalização implica não apenas em adaptar conteúdo ao indivíduo, mas também em como o conteúdo é comunicado para efeito máximo.
- • Integração pedagógica - diversas formas de suporte ao aprendizado devem ser disponibilizadas de maneira integrada, tais como tutores inteligentes, ambientes exploratórios e de simulação, autoria e comunicação em grupo, tutoriais, além de mecanismos para cooperação.
- • Integração funcional - a adição incremental de componentes funcionais deve ser possível e, para isso, devem ser explorados padrões que permitem interoperabilidade; metodologias para definição, projeto e implementação de componentes tornam-se obrigatórias (Roschelle *et al.*, 1999).
- • Acessibilidade - o acesso pelos diferentes atores deve se realizar através da *Web*; isto significa que o ambiente colaborativo de aprendizado estará disponível na Internet, permitindo acesso e integração de recursos disponíveis que sejam de interesse no processo de aprendizado.
- • Reuso - os resultados obtidos devem poder ser aproveitados não somente no contexto do aprendizado de programação, mas em outros contextos de aprendizado com características similares.
- • Meta cognição - o estudante necessita realizar reflexões sobre o processo e resultados do seu aprendizado, para isso deve poder monitorar e regular suas atividades relacionadas à solução de problemas (Collins *et al.*, 1989).

A proposta de arquitetura para o ambiente pretendido pode ser apresentada a partir de diferentes perspectivas, como se segue.

3.1. Tecnologias de suporte

O mapa de tecnologias, apresentado na figura 1, procura indicar que há mecanismos disponíveis que podem ser adaptados e combinados para oferecer os meios necessários ao aprendizado colaborativo, no domínio considerado. O ambiente alvo deverá ser constituído pela tecnologia encontrada na interseção escura da figura. Ou essa tecnologia já existe e necessita ser identificada, ou terá que ser desenvolvida.

Entre as tecnologias disponíveis para Acesso e Localização de Informação encontram-se: sistemas de bancos de dados, relacionais e orientados a objetos; linguagens de consulta, tais como SQL, OQL, QBE e linguagens gráficas; busca e recuperação de informação; extração e integração de informação; padrões para estruturação, modelagem e comunicação de objetos educacionais, tais como HTML, XML, IMS, ADL e AICC; hipertexto e hipermídia; e criação e manutenção de ontologias.

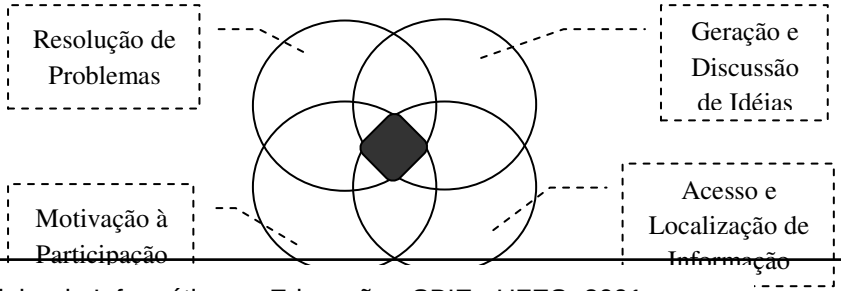

Fig. 1 - Mapa de tecnologias

Para a geração e discussão de idéias, há um amplo leque de tecnologia desenvolvida que pode ser usada para suportar o trabalho cooperativo suportado por computadores (*Computer Suported Cooperative Work - CSCW*) e a tomada de decisão em grupos (*Group Decision Support Systems).*

Para a resolução de problemas pode-se contar com um amplo leque de editores (para indivíduos e grupos), compiladores e depuradores (*debuggers*) de programas.

Alguns elementos extraídos dos textos sobre os chamados Sistemas Tutores Inteligentes Baseados em Diálogo (Graesser *et al*., 2000) reforçam a idéia de que uma interface em língua natural motivaria a participação dos aprendizes, pois o diálogo conversacional desempenha um papel importante no aprendizado. Desta forma, sistemas tutores devem usar processamento de língua natural, para se tornarem mais efetivos no encorajamento ao aprendizado aprofundado.

3.2. Níveis de abstração

A visão em níveis do ambiente alvo (figura 2) pode ser dividida em quatro camadas: i) apresentação, ii) operações para aprendizado, iii) mecanismos de suporte e iv) infra-estrutura computacional. O nível Apresentação constitui a interface entre atores e ambiente, através da qual dados são exibidos e percebidos. Este nível é constituído por dois subníveis: navegador (*browser*) e componentes de percepção, como em (Ebner *et al*., 2000), que são realizações dos componentes manipulados no nível Operações para Aprendizado, ou seja, representações perceptíveis das abstrações computacionais que constituem objetos educacionais ou resultados das funcionalidades disponibilizadas.

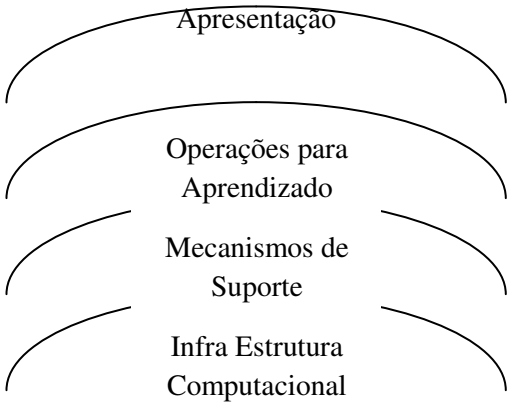

Fig. 2 - Níveis do ambiente alvo

No nível Mecanismos de Suporte, encontram-se ferramentas para o suporte aos diversos requisitos do ambiente, tais como colaboração, comunicação, extração de informação, descoberta de serviços, personalização, meta cognição, entre outros.

No nível Infra-estrutura Computacional encontram-se facilidades básicas para comunicação, gerenciamento de armazenamento persistente e processamento de dados.

3.3. Módulos Funcionais

Os módulos funcionais que deverão compor o ambiente são mostrados na figura 3.

A base de objetos educacionais em conjunto com o Processador de Consultas devem oferecer a possibilidade de relacionar objetos educacionais para as mais variadas finalidades.

Para acesso à informação na *Web*, além do Processador de Consultas, há o Processador de Informação para extração, tratamento semântico, validação, transformação, classificação, filtragem e mineração de dados. *Wrappers* e agentes são usados para descoberta e gerência de informação nas diversas fontes da *Web*. Existe uma base com meta dados para as fontes, regras de extração, dicionários semânticos, regras de autenticação e validação, entre outras.

Para manipulação de programas, o estudante lança mão de editores e visualizadores de objetos, através dos quais pode ativar compiladores, depuradores ou simuladores. Para comunicação, ferramentas para Listas, *e-mail* e *chat* estão disponíveis, suportando texto, voz e imagem, além disso, existem módulos para suporte à colaboração.

Alguns módulos são explorados com mais detalhes a seguir, devido ao seu papel central no sistema, sendo utilizados em várias etapas do processo de solução de problemas.

A função do Verificador de Semelhança de Programas (VSP) é comparar programas indicando o grau de semelhança entre os mesmos. Após receber as submissões individuais, o VSP pode ser acionado para tentar encontrar indícios de plágio. Na fase de colaboração, o VSP permitirá identificar diferenças significativas entre as soluções apresentadas, possibilitando estimular a interação entre os autores, a fim de que elas possam ser contrastadas e discutidas. O VSP permite também buscar na base de soluções programas semelhantes ao apresentado, de forma que os autores podem apreciar os comentários feitos a soluções semelhantes desenvolvidas anteriormente. Isto deve ser feito quando da submissão final, mas pode também acontecer quando da submissão inicial.

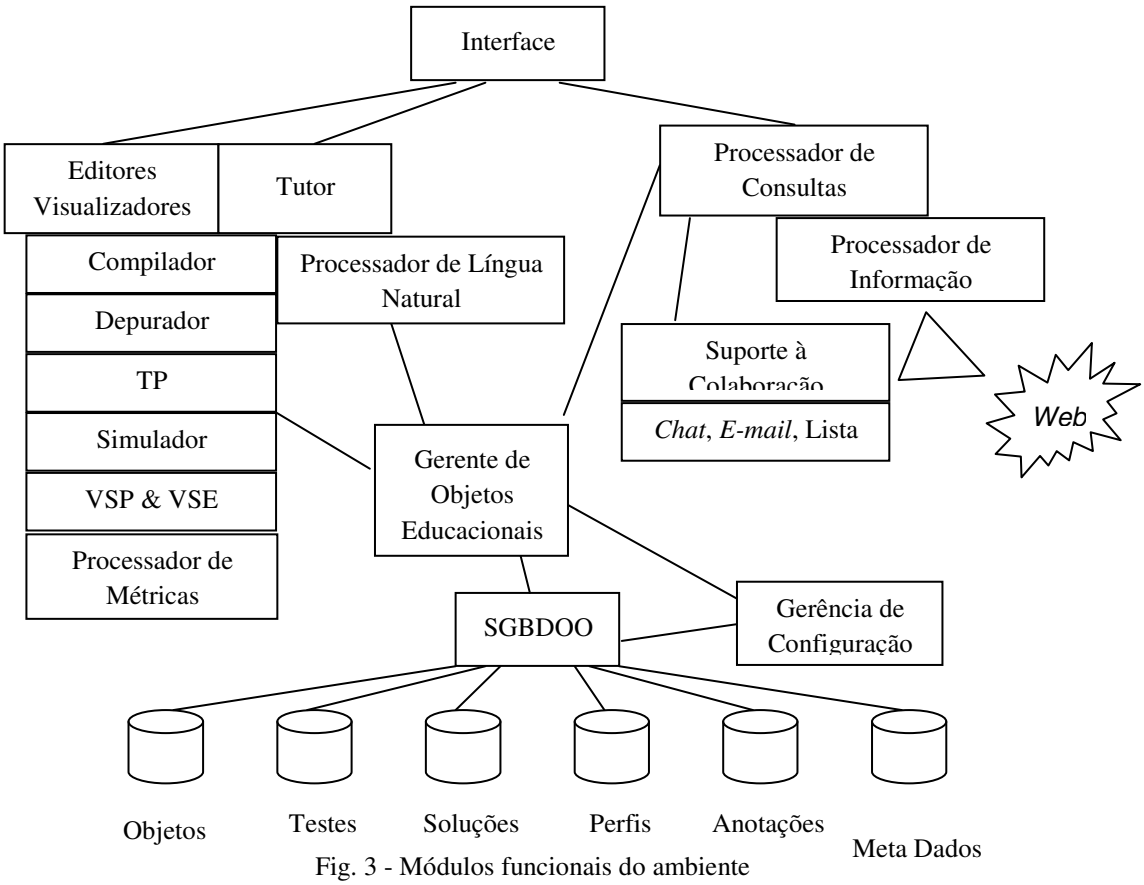

Fig. 3 - Módulos funcionais do ambiente

O módulo Verificador de Semelhança de Explicações (VSE) tem papel análogo ao de verificação de semelhança de programas, mas compara os textos explicativos dos programas. Pode ser usado tanto para comparar as explicações de vários estudantes entre si, como para comparar as soluções dos estudantes às soluções armazenadas na base de soluções.

Para a construção do módulo VSE, podem ser usados métodos de processamento estatístico de textos, desenvolvidos nas área de recuperação e filtragem de informação textual (Baeza-Yates e Ribeiro-Neto, 1999) e mineração de textos (Hearst, 1999), assim como eventualmente abordagens mais sofisticadas de processamento de língua natural.

Após compilar o programa, o testador submete-o a um conjunto de testes do tipo caixa preta, visando verificar a funcionalidade básica do sistema. O módulo Testador de Programas (TP) pode ser construído a partir de pacotes disponíveis publicamente, como o DejaGnu (Savoye, 1996) e o Praktomat (Zeller, 2000).

O módulo Tutor será baseado em uma interface em língua natural, com o intuito de propiciar um relacionamento mais "humano" entre o sistema e os estudantes. Também para alcançar uma eficiência maior no processo de aprendizagem. Para isso, deve se basear em sistemas existentes, com resultados satisfatórios, como o AutoTutor (Freedman *et al.*, 2000), o CIRCSIM-Tutor (Glass, 2000) e o Atlas (Rosé, 2000).

Os tutores baseados em diálogo freqüentemente se referem à comunicação escrita, ou seja, há a necessidade da elaboração de um *parser* para a língua usada. O *parser* utiliza uma base de conhecimento sobre o assunto da disciplina que se deseja ensinar (programas, textos de programação, etc.) e que deve estar atualizada em relação ao progresso da aprendizagem. Isto significa que o sistema deve ter a capacidade de adaptação ao usuário. Ferramentas da Inteligência Artificial, com capacidade de aprendizagem e generalização, como as redes neurais artificiais, podem ser usadas (Olde *et al.*, 1999). Métodos estatísticos para extração de conhecimento lingüístico também podem ser usados, como a Análise Semântica Latente (Wiemer-Hastings, 1999). Conhecimento da língua e de teorias lingüísticas são úteis na implementação de *parsers* conexionistas psicolingüisticamente plausíveis (Miikkulainen, 1996).

4. Considerações Finais

A proposta de um ambiente colaborativo para aprendizado de programação visa contribuir para a superação de dificuldades verificadas, decorrentes do contexto atual de ensino de programação nas disciplinas introdutórias nos cursos da área de computação. O objetivo deste trabalho é apresentar uma arquitetura para esse tipo de ambiente, baseada em pressupostos pedagógicos, que ofereça meios para geração e discussão de idéias, resolução de problemas, acesso e localização de informação *on-line* útil e motivação à participação dos estudantes.

Neste momento, está sendo finalizada a definição geral da arquitetura do sistema, estando disponíveis protótipos para alguns dos módulos, como por exemplo um simulador gráfico para manipulação de ponteiros na linguagem C. Outros módulos encontram-se em implementação, entre eles o módulo verificador de semelhança de programas, que usa o algoritmo RKS-GST (Wise, 1996), desenvolvido com o propósito de identificar possíveis ocorrências de plágios de programas.

As diretrizes estabelecidas para a implementação do ambiente, em especial aquelas relacionadas com a engenharia de *software* (integração funcional, acessibilidade e reuso), orientam a escolha de linguagens de programação, padrões para representação de objetos educacionais e documentos, além de mecanismos de suporte e infra estrutura computacional.

Outro esforço em andamento diz respeito à busca por ferramentas e componentes de *software* existentes, que possam ser usados como base para a construção de alguns dos módulos identificados.

5. Referências

BAEZA-YATES, R.A., RIBEIRO-NETO, B., "Modern Information Retrieval", ACM Press, 1999.

COLLINS, A., BROWN, J.S. AND NEWMAN, S.E., "Cognitive apprenticeship: teaching the crafts of reading, writing, and mathematics". In. L.B. Resnick, ed., Knowing, learning, and instruction: essays in honor of Robert Glaser. Hillsdale, NJ: Erlbaum, 1989.

COMPUTER SOCIETY CONNECTION, "Defining Computing Curricula for the Modern Age", IEEE Computer, 34(6):75-77, June, 2001.

CONSTANTINO-GONZÁLEZ, M.A. SUTHERS, D.D., "A Coached Collaborative Learning Environment for Entity-Relationship Modeling". In Intelligent Tutoring Systems, Proceedings of the 5th International Conference (ITS 2000), G. Gauthier, C. Frasson and K. Van Lehn (Eds.), pp. 325-333. Springer-Verlag. 2000.

EBNER, E., SHAO, W. AND TSA, W., "The Five-Module Framework for Internet Application Development", ACM Computing Surveys, 32(1es), march, 2000.

FREEDMAN, R., ALI, S.S., MCROY, S., "What is an Intelligent Tutoring System?" ACM Intelligence, Fall, 15-16. 2000.

GLASS, M., "Processing Language Input in the CIRCSIM-Tutor Intelligent Tutoring System". AAAI 2000 Fall Symposium on Building Dialogue Systems for Tutorial Applications. 2000.

GOKHALE, A.A., "Collaborative Learning Enhances Critical Thinking", Journal of Technology on Education, 7(1), 1995.

GRAESSER, A.C., ROSÉ, C.P., JORDAN, VANLEHN, K., "Intelligent Tutoring Systems with Conversational Dialogue". 2000. http://www.pitt.edu/~circle/Resources.html

HEARST, M.A., "Untangling Text Data Mining", Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics, 1999.

LITE - LABORATÓRIO INTERDISCIPLINAR DE TECNOLOGIAS EDUCACIONAIS, "Integrando o Pedagógico e o Tecnológico", Faculdade de Educação, Unicamp, 2000. http://www.unicamp.br/~hans/lite/ sapiens/relat.html

MEC/SESu, "Home Page da Comissão de Especialistas de Ensino de Computação e Informática", 1997. http://www.inf.ufrgs.br/mec

MIIKKULAINEN, R., "Subsymbolic Case-role Analysis of Sentences with Embedded Clauses", Cognitive Science 20, 47-73. 1996.

OLDE, B.A., HOEFFNER, J., CHIPMAN, P., GRAESSER, A.C., "A Connectionist Model for Part of Speech Tagging". Proceedings of the American Association for Artificial Intelligence, Menlo Park, CA: AAAI Press, 172-176. 1999.

PEDNAULT, E.P.D., "Representation is Everything", CACM, 43(8):80-83, August, 2000.

PROULX, V.K., "Programming Patterns and Design Patterns in the Introductory Computer Science Course", Proc. of the 31st SIGCSE, pp. 7-12, Auxtin, TX, USA, march, 2000.

RASALA, R., PROULX, V., FELL, H.J. AND RAAB, J., "Computer Science Education Research Group", 2000. http://www.ccs.neu.edu/teaching/EdGroup/

ROSCHELLE, J., DIGIANO, C., KOUTLIS, M., REPENNING, A., PHILLIPS, J., JACKIW, N., SUTHERS, D., "Developing Educational Software Components", IEEE Computer, 32(9):50-58, September, 1999.

ROSÉ, C.P., "Facilitating the Rapid Development of Language Understanding Interfaces for Tutoring Systems". Proceedings of the AAAI Fall Symposium on Building Tutorial Dialogue Systems. 2000.

SAVOYE, R., "The DejaGnu Testing Framework". Free Software Foundation, Inc. January, 1996.

SOLOWAY, E., "Learning to Program = Learning to Construct Mechanisms and Explanations". Com. Of the ACM, 29(9). September, 1986.

SUTHERS, D.D., "Computer Aided Education and Training Initiative", Technical Report, Learning Research and Development Center, University of Pittsburgh, 1998.

WIEMER-HASTINGS, P., "How Latent is Latent Semantic Analysis?" in Proceedings of IJCAI'99 – Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden, July 31-August 6, Volume 2, Morgan Kaufmann, 932-937. 1999.

WISE, M.J., "YAP3: Improved Detection of Similarities in Computer Program and Other Texts". ACM SIGCSE'96. 1996.

ZELLER, A., "Making Students Read and Review Code". ACM ITICSE 2000.