UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

VHS-AM: EXTENSÃO DO VISUAL STUDIO .NET PARA A INTEGRAÇÃO DE HASKELL COM A FERRAMENTA ASSIGNMENT MANAGER

TRABALHO DE GRADUAÇÃO

Aluno: André Wilson Brotto Furtado (awbf@cin.ufpe.br)
Orientador: André Luís de Medeiros Santos (alms@cin.ufpe.br)

Abril de 2004

ASSINATURAS

Este Trabalho de Graduação é resultado dos esforços do aluno André Wilson Brotto Furtado, sob a orientação do professor André Luís de Medeiros Santos, na participação do projeto VHS-AM, conduzido pelo Centro de Informática da Universidade Federal de Pernambuco e aprovado pela *Microsoft Shared Source Initiative*. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

André Wilson Brotto Furtado

André Luís de Medeiros Santos

RESUMO

A linguagem funcional Haskell é consolidada como uma das principais linguagens em áreas que empregam o paradigma de programação funcional. Entretanto, é reconhecido que algumas limitações provocam uma sub-utilização do potencial da linguagem. A timidez na integração com ferramentas de mercado e a ausência de ambientes de desenvolvimento, objetivando tornar a etapa de programação mais produtiva e menos sujeita a erros, são duas das principais restrições identificadas. O trabalho apresentado neste documento é um subconjunto do projeto VHS-AM, que pretende oferecer um ambiente de programação que dê suporte à utilização prática e produtiva de Haskell, através do modelo de extensão e automação oferecido pela IDE Visual Studio .NET 2003. Adicionalmente, é validada a integração da IDE estendida com a ferramenta acadêmica Assignment Manager, com o objetivo de disseminar o uso de Haskell e ampliar seu o campo de aplicação. Por fim, é realizada uma discussão em torno da utilização da metodologia de desenvolvimento de software Pro.NET na condução formal do projeto.

Palavras-chave:
Haskell, integração, extensão, Visual Studio .NET, Assignment Manager, Pro.NET.

ABSTRACT

Haskell is one of the most popular programming languages in areas that use the functional programming paradigm. However, it is known that some language constraints make Haskell to be overlooked in other major programming areas. Such limitations include the loose integration with commonly used tools and the lack of an integrated development environment, which would lead to a more productive and less error-prone programming process. The work presented in this paper is a subset of the VHS-AM project, which intends to offer, to the Haskell community, an integrated development environment that supports a practical and productive use of the Haskell language. The approach is based on Microsoft Visual Studio .NET 2003 extensibility and automation model. In addition, the extended IDE is validated against a Microsoft academic tool called Assignment Manager, aiming at spreading around the utilization of Haskell and expanding its application boundaries. Finally, this work discusses the usefulness of the Pro.NET software development methodology as a means to formally conduct the VHS-AM project.

Keywords:
Haskell, integration, extensibility, Visual Studio .NET, Assignment Manager, Pro.NET.

AGRADECIMENTOS

Agradecer é correr o risco de esquecer alguém. Àqueles que não foram listados aqui, peço profundas desculpas e espero que compreendam que tal injustiça não foi causada por vontade própria, mas por traição da memória.

À minha família, pais e irmãs, não há palavras que sejam suficientes. Vocês sabem, mais do que eu mesmo, que tudo isto se deve a vocês. Obrigado por todo o apoio, seja ele do quarto ao lado, do estado ao lado ou do continente ao lado.

À Lícia, de quem o Word insiste em tirar o acento, vai um pacote conjunto de agradecimentos e desculpas. Obrigado pela palavra, pelo silêncio e pelos momentos, nas horas certas, que tornaram o caminho até aqui tão mais leve.

Ao amigo e professor André Santos, é hora de agradecer não apenas pela realização deste trabalho, como também por todas as oportunidades ao longo de uma longa graduação. Obrigado não apenas pelo empenho, mas também pela confiança e motivação durante todo esse tempo.

Obrigado a toda a turma CC99.1, pelo companheirismo e amizade que nos ajudou a chegar até aqui. Em especial, agradeço a André Amaral, Fernando e Gustavo (vulgos Coelho, Passaro e Danzi) por todos os momentos, projetos e discussões pelas quais passamos juntos. Agradeço também a Igor Gatis pelas horas e horas de *MSN Messenger* compartilhadas durante a realização deste projeto, madrugada após madrugada.

A Mauro Araújo e Marden Menezes, agradeço a oportunidade de estar trabalhando com eles neste projeto, por poder compartilhar não apenas todas as conquistas como também todas as dificuldades. Apenas para não perder o costume... vocês já deram uma olhada nas suas pendências? ☺

A todos do Centro XML / Qualiti, fica o agradecimento pelos recursos disponibilizados, assim como pela a flexibilidade e bom senso essenciais para a conclusão deste trabalho. A Adeline, um agradecimento especial pelo apoio de sempre e as velhas discussões técnicas.

Aos amigos dos tempos de colégio, obrigado por ainda estarem presentes, pela disposição sempre renovada a cada encontro e a cada lembrança.

Simon e Krasimir... será que algum dia vocês vão ler (e entender) isso? Caso sim, agradeço pela cooperação e ajuda, sem as quais este trabalho não seria possível (ou seria bem mais complexo), assim como pela paciência nos momentos difíceis.

Por fim, agradeço ao Google, a maior invenção da Web, por todas as buscas proporcionadas. As coisas seriam realmente mais difíceis sem você...

André W. B. Furtado, 07/04/2004

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. INTRODUÇÃO

Integração: esta é a melhor palavra para resumir não apenas o produto final construído através deste este projeto, como também seu próprio processo de desenvolvimento. Este documento procura estabelecer uma ordem, cronológica e funcional, para situar cada uma das tecnologias, ferramentas, conceitos, atividades e equipes, integrados, conjuntamente, na realização do projeto VHS-AM e, conseqüentemente, na realização deste Trabalho de Graduação. É importante deixar claro, já a partir deste momento, que o projeto VHS-AM é um super-conjunto deste Trabalho de Graduação, sendo apresentado, ainda nesta seção, um relacionamento mais detalhado entre ambos.

Grosso modo, o objetivo principal do projeto VHS-AM consiste em estender um ambiente integrado de desenvolvimento, o Visual Studio .NET 2003, para que o mesmo ofereça suporte à linguagem funcional Haskell. Adicionalmente, também faz parte do escopo do projeto a integração desse ambiente estendido, intitulado Visual Haskell Studio (ou **VHS**), com a ferramenta acadêmica Assignment Manager (ou simplesmente **AM**), que gerencia, para professores e alunos, o processo de publicação e submissão de tarefas acadêmicas que envolvem codificação. Deste modo, foi escolhido "**VHS-AM**" como codinome para o projeto, indicando que seu produto final integrará, portanto, três diferentes componentes: a linguagem Haskell, a IDE[1] Visual Studio .NET 2003 e a ferramenta Assignment Manager.

O tipo de suporte a ser oferecido pelo Visual Studio .NET 2003 à linguagem Haskell (suporte à edição, suporte a projetos na IDE, etc.), assim como um maior detalhamento da ferramenta Assignment Manager e do próprio Visual Studio .NET 2003, serão apresentados no decorrer deste trabalho. Esta seção, por sua vez, está focada em apresentar, em linhas gerais, o que levou ao surgimento do projeto VHS-AM e discutir a relação do mesmo com este Trabalho de Graduação.

De modo a evitar que o texto aqui apresentado se torne muito carregado, o restante deste documento poderá se referir ao Visual Studio .NET 2003 como simplesmente "VS.NET". Analogamente, poderá ser utilizada a sigla "AM" para realizar referências à ferramenta Assignment Manager.

---

[1] IDE (*Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento) é um tipo de programa que oferece suporte à etapa de implementação de aplicações, da edição de código às tarefas de compilação e depuração, aumentando a produtividade do desenvolvedor.

1.1 A Origem do Projeto VHS-AM

Em um dado momento, a idéia original para este Trabalho de Graduação consistia apenas na integração do VS.NET com a linguagem funcional Haskell, o que seria realizado em parceria com a *Microsoft Research Cambridge* [51]. Entretanto, devido a uma oportunidade captada através da *Microsoft Shared Source Initiative* e de um *request for proposals* (RFP) da própria Microsoft, foi possível inserir este Trabalho de Graduação em um contexto maior, tornando-o, portanto, um subconjunto do projeto VHS-AM. Esse processo de evolução do projeto é apresentado em mais detalhes nesta subseção.

1.1.1 O Projeto VHS Original

Alguns meses antes da aprovação da proposta do projeto VHS-AM, o orientador deste Trabalho de Graduação já havia estabelecido contato com outros pesquisadores da *Microsoft Research Cambridge* (UK), no sentido de planejar uma futura cooperação entre desenvolvedores para construir uma extensão do Visual Studio .NET 2002[2] para o suporte à linguagem Haskell. Este projeto foi formalmente estabelecido em meados do segundo semestre de 2003, com o codinome VHS (*Visual Haskell Studio*), tendo à frente o pesquisador Simon Marlow [52], da própria *Microsoft Research Cambridge*. As atividades iniciais do aluno envolvido neste Trabalho de Graduação, portanto, consistiram em estudar, analisar e testar o trabalho até então desenvolvido no projeto VHS. O mesmo também foi responsável por consistir o status e os objetivos do projeto em um *website* (acessível em http://www.cin.ufpe.br/~haskell/vhs) sendo criada uma identidade visual para o projeto, apresentada na Figura 1. Deste modo, o aluno foi integrado ao projeto VHS, juntando-se à equipe inicialmente formada por seu orientador (professor André Santos), o pesquisador Simon Marlow e o desenvolvedor búlgaro Krasimir Angelov.

Figura 1 – Identidade visual para o projeto VHS

[2] Na época, a versão 2002 do Visual Studio .NET ainda era a mais popular entre os desenvolvedores, sendo a versão 2003 uma novidade.

1.1.2 A *Microsoft Shared Source Initiative*

De modo a disponibilizar seu código fonte de maneira mais ampla para clientes, parceiros, o governo e, principalmente, para a academia, a Microsoft criou uma série de programas e licenças especiais para atingir tal objetivo, obtendo também benefícios a partir deles. Este conjunto de licenças e programas chama-se *Microsoft Shared Source Initiative* [58].

Um dos casos mais bem sucedidos da *Microsoft Shared Source Initiative*, por exemplo, consiste nos *ASP.NET Starter Kits* [54], utilizados pela empresa para a difusão da tecnologia ASP.NET [64]. Disponíveis inclusive a partir da MSDN Brasil [68], estes kits são exemplos de aplicações ASP.NET onde é possível encontrar códigos (completos, documentados e gratuitos) que auxiliam no desenvolvimento de tarefas comuns na Web, podendo ser estendidos, inclusive, para propósitos comerciais.

No caso do VHS-AM, o programa da *Microsoft Shared Source Initiative* que viabilizou o projeto foi o *Visual Studio .NET Academic Tools Source Licensing Program* [65]. Este programa permite que estudantes, professores e pesquisadores acadêmicos, entre outros indivíduos, possam ter acesso e alterar o código-fonte do conjunto de componentes que formam a ferramenta Assignment Manager.

1.1.3 RFP para Extensão do Assignment Manager

No segundo semestre de 2003, a Microsoft anunciou mundialmente uma “chamada por propostas” (termo mais comumente conhecido como *request for proposals*, ou RFP) para a extensão da ferramenta Assignment Manager, cujo código é acessível através da *Microsoft Shared Source Initiative*, como explicado acima.

O aluno deste Trabalho de Graduação ofereceu sugestões para a proposta inicialmente elaborada pelo seu orientador, em resposta ao RFP, contribuindo, portanto, para a versão final da mesma. Em resumo, a proposta enviada consistiu na extensão da ferramenta Assignment Manager para que a mesma suportasse projetos desenvolvidos na linguagem Haskell, além de alterações no próprio VS.NET que também objetivassem o suporte a Haskell. Adicionalmente, a proposta incluiu a especificação de uma equipe de 5 integrantes para a realização do projeto, sobre a qual o aluno deste Trabalho de Graduação deveria assumir, além das originais tarefas de desenvolvimento, atividades de coordenação e gerência de projeto. O aluno seria, portanto, o responsável pela condução formal do projeto através de uma metodologia de desenvolvimento de software bem-definida: a Pro.NET[3] [14].

---

[3] A Pro.NET é uma metodologia desenvolvida pelo Centro de Tecnologia XML do Recife [63], baseada no *Microsoft Solutions Framework* [86]. Ela é apresentada em detalhes no Capítulo 6.

Ainda no segundo semestre de 2003, a Microsoft anunciou a proposta enviada pela UFPE como uma das cinco vencedoras da RFP [50]. Os recursos obtidos com esta aprovação permitiram a consolidação da equipe inicialmente proposta: os alunos Mauro La-Salette Araújo e Marden Menezes Costa, ambos da graduação do curso de Ciências da Computação do Centro de Informática da UFPE, foram integrados à equipe de desenvolvimento, assim como o professor Paulo Henrique M. Borba, que se tornou um dos responsáveis pela homologação final do projeto, juntamente com o professor André Santos.

No dia 15 de dezembro, portanto, foi iniciado oficialmente o projeto VHS-AM. Do mesmo modo que o projeto VHS original, foi criado um *website* para o VHS-AM (acessível a partir de www.cin.ufpe.br/~haskell/vhs-am), assim como uma identidade visual para o mesmo, apresentada na Figura 2.

Figura 2 – Identidade visual para o projeto VHS-AM

Como é possível perceber, o projeto VHS-AM possui uma importante interseção com o projeto VHS original, havendo se estabelecido o desafio de combinar e sincronizar os esforços dos dois projetos para garantir a satisfação dos objetivos de ambos da maneira mais produtiva possível.

1.2 O Trabalho de Graduação no Contexto dos Demais Projetos Apresentados

Observou-se que a participação do aluno nos projetos VHS e VHS-AM reunia os requisitos necessários à realização de um Trabalho de Graduação para o CIn-UFPE, dentre os quais é possível citar:

- **Pesquisa sobre o estado-da-arte** referente aos mecanismos de extensão do Visual Studio .NET para o suporte a novas linguagens;
- **Atividades não-triviais de implementação** demandadas pela integração de diferentes conceitos (programação funcional, noções de compiladores, etc.) ferramentas (VS.NET, AM, GHC[4], etc.) e tecnologias (componentes COM, modelo de extensão do VS.NET, etc.);

[4] O GHC, ou Glasgow Haskell Compiler [37], é o mais popular compilador para a linguagem Haskell, sendo apresentado em mais detalhes ao longo deste documento.

- Contribuição para a **realização de um trabalho teórico** referente à análise da adequação da recente metodologia Pro.NET a um projeto externo ao Centro de Tecnologia XML, experiência inédita até então.

Desse modo, com o amadurecimento e submissão da proposta deste Trabalho de Graduação, o aluno passou a estar envolvido em três projetos bastante correlacionados: o projeto VHS, o projeto VHS-AM e o próprio Trabalho de Graduação (TG).

Uma organização comparativa entre esses três projetos, em relação ao tempo, pode ser visualizada através da Figura 3.

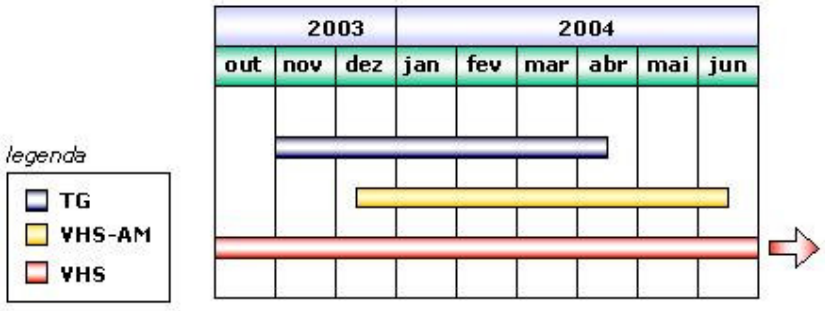

Figura 3 – Organização dos três projetos em relação ao tempo

Esta organização dos projetos no tempo revela que:

- Este Trabalho de Graduação e o projeto VHS-AM possuem prazos bem-definidos, ao contrário do projeto VHS, que é um projeto da comunidade Haskell como um todo e pode vivenciar melhorias contínuas sem compromissos temporais;
- Após a finalização do prazo deste Trabalho de Graduação, o aluno ainda estará envolvido diretamente com os projetos VHS-AM e VHS, sendo esperado que o conjunto de resultados apresentado neste documento seja enriquecido e melhorado no futuro.

Uma organização comparativa dos projetos em relação ao escopo, por sua vez, pode ser visualizada na Figura 4.

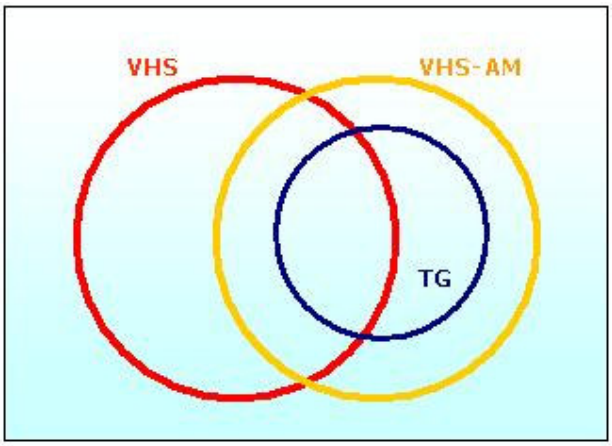

Figura 4 – Organização dos três projetos em relação ao escopo

No tocante ao escopo dos três projetos, pode-se constar que:

- As atividades executadas pelo aluno neste Trabalho de Graduação, na verdade, são um subconjunto do projeto VHS-AM. Este subconjunto se refere às atividades realizadas pelo aluno no projeto VHS-AM até o prazo de entrega deste Trabalho de Graduação, isto é, meados de abril de 2004;
- O projeto VHS-AM possui uma (importante) interseção com o projeto VHS. Entretanto, existem atividades específicas para cada projeto: o projeto VHS não possui, necessariamente, nenhum compromisso com a ferramenta Assignment Manager, enquanto o projeto VHS-AM não pode assumir compromissos relativos à conclusão da implementação de novas funcionalidades que eventualmente sejam planejadas no projeto VHS. Em outras palavras, o planejamento decorrente do uso da metodologia Pro.NET está restrito à equipe do projeto VHS-AM;
- Algumas atividades executadas pelo aluno satisfazem a todos os três projetos, enquanto outras são compartilhadas apenas pelo projeto VHS-AM e este Trabalho de Graduação.

Essas informações apresentadas, referentes à modelagem da correlação entre os três projetos, mostram-se não apenas necessárias para um melhor entendimento deste trabalho, como também se revelaram essenciais para que os próprios desenvolvedores dos projetos tivessem uma noção clara sobre suas fronteiras de atuação. Isto permitiu um trabalho mais consciente e, conseqüentemente, mais produtivo.

É importante estar claro ao leitor que a utilização do termo “este projeto”, no restante deste documento, se referirá ao Trabalho de Graduação e conseqüentemente ao projeto VHS-AM, devido ao fato do primeiro estar inserido no segundo. As atividades realizadas por um outro desenvolvedor, por sua vez, serão devidamente creditadas ao mesmo, de modo a evitar a atribuição indevida de resultados, obtidos nos projetos VHS e VHS-AM, ao aluno deste Trabalho de Graduação.

1.3 Organização deste Documento

O restante deste relatório está estruturado da seguinte maneira: o Capítulo 2 apresenta o contexto tecnológico que motivou e embasou o desenvolvimento deste trabalho, assim como trabalhos similares e anteriores. O Capítulo 3, por sua vez, apresenta o escopo do projeto em termos de seus principais casos de uso. O Capítulo 4 discute os diferentes mecanismos de extensão do VS.NET à luz dos casos de uso especificados, enquanto o Capítulo 5 detalha a abordagem de implementação e apresenta os resultados obtidos com este Trabalho de Graduação, o que se reflete no atual *status* do projeto VHS-AM. O Capítulo 6 apresenta a metodologia Pro.NET e como a mesma foi aplicada para o gerenciamento do projeto, enquanto o Capítulo 7 conclui acerca do trabalho, apontando os próximos passos a serem realizados. A lista das Referências Bibliográficas utilizadas na elaboração deste documento, por fim, encerra o trabalho.

2. CONTEXTO

Esta seção tem como objetivo inserir o leitor no contexto da realização deste trabalho, permitindo um melhor entendimento da motivação e relevância do mesmo. Serão apresentados os conceitos, ferramentas e tecnologias necessárias ao desenvolvimento do projeto VHS-AM, assim como alguns trabalhos similares e anteriores que influenciaram seu desenvolvimento.

2.1. Uma Breve História das Linguagens de Programação

Quando o ser humano passou a manusear aparelhos mais complexos do que o ábaco e as arcaicas "máquinas somadoras" para computar seus cálculos [24], tornou-se evidente a necessidade pela automação do procedimento a ser executado pelos instrumentos que um dia evoluiriam até o hardware da atualidade. Surgiram, então, os cartões perfurados, fios elétricos e válvulas que, algum tempo depois, foram substituídos por um procedimento um pouco mais automatizando de programação: a escrita de código binário ou hexadecimal capaz de ativar e desativar transistores. Não demorou muito para que este método primitivo de programação, efetuado instrução por instrução, fosse considerado uma atividade entediante, demorada e bastante sujeita a erros [71]. Adicionalmente, os programas elaborados eram bastante difíceis de ler e ainda mais complexos de se manter, principalmente pelo endereçamento absoluto de memória.

O desenvolvimento de linguagens de baixo nível, como as diferentes versões de Assembly, chegou a reduzir o problema, mas de maneira bastante limitada. Os programadores ainda desejavam um maior poder de abstração (como estruturas de código, pontos flutuantes e indexação de *arrays*), sendo necessária, portanto, uma separação maior entre as tarefas especificadas por um programa e os passos executados pelo hardware.

O surgimento dos compiladores, na década de 50, viabilizou a concretização de tal desejo [8]. As primeiras linguagens de médio-nível, como Fortran (1957), Lisp (1958) e Cobol (1959), foram aos poucos dividindo espaço com as linguagens Assembly [46]. Hoje, meio século depois, cerca de 2500 linguagens de programação estão catalogadas [45], sendo esta variedade constituída das populares C, C# e Java às praticamente desconhecidas Pizza, TESSARACAT e Woodenman.

Entre essas milhares de linguagens de programação, há uma que se destaca por fazer parte das principais tecnologias utilizadas em áreas que empregam o paradigma de programação funcional: a linguagem Haskell [34]. Apesar de ser bastante difundida neste nicho específico, a linguagem funcional Haskell enfrenta atualmente algumas limitações que impedem sua maior popularização e o aumento de seu campo de aplicação [80].

2.2 Compreendendo Linguagens Funcionais: Virtudes e Deficiências

C, Java e Pascal são linguagens de programação imperativas[5], no sentido de que programas construídos nessas linguagens consistem em uma seqüência de comandos, executados estritamente um após o outro. Um programa feito em uma linguagem funcional como Haskell[6], por outro lado, é uma única expressão. A execução de um programa funcional é, na verdade, a própria avaliação da expressão .

Uma planilha eletrônica, em que o valor de uma célula é computado em termos de outras células, exemplifica bem o conceito de linguagens funcionais [38]. Em programas deste tipo, não é especificada a ordem em que as células são calculadas; espera-se que a planilha eletrônica compute as células em uma ordem que respeite suas dependências. Além disso, não é dito ao programa como e quando a memória será alocada; espera-se que a planilha, apesar de exibir um plano aparentemente infinito de células, aloque memória apenas para as células em uso. Por fim, o valor de uma célula é especificado por uma expressão, e não por uma seqüência de comandos que computa o seu valor.

Uma das principais características de linguagens funcionais, portanto, consiste na abstração oferecida ao programador, que interage com a linguagem especificando em alto-nível o que deve ser feito, e não como fazer. Linguagens imperativas, por outro lado, procuram atenuar o problema da menor intuitividade de programação, provocada pela seqüencialização de comandos, através da introdução de novas palavras chaves (*while*, *for*, *foreach*, etc.) ou de bibliotecas (APIs[7]). Entretanto, linguagens imperativas foram concebidas pela filosofia da seqüencialização de comandos, impedindo que esse conceito seja completamente eliminado neste paradigma de linguagem. Desse modo, linguagens imperativas não conseguem atingir o mesmo nível de abstração proporcionado por linguagens funcionais [76].

A utilização de linguagens funcionais também oferece alguns dos principais benefícios encontrados na programação orientada a objetos, paradigma reconhecido como um dos

---

[5] Alguns autores consideram linguagens orientadas a objeto, como Java e C#, um subconjunto das linguagens imperativas. Outros autores, entretanto, preferem separar os dois conceitos. Neste trabalho, será considerada a primeira abordagem, ressaltando-se que a segunda não está tecnicamente incorreta.

[6] Outros exemplos de linguagens funcionais são a pioneira Lisp [47], Caml [13], Clean [73], Standard ML (SML) [72] e Erlang [18], sendo esta última a linguagem adotada pela empresa Ericsson em seu processo de desenvolvimento interno.

[7] Uma API (*Application Program Interface*) é um conjunto de rotinas e protocolos reutilizados para o desenvolvimento mais produtivo de aplicações. APIs são consideradas verdadeiros "blocos de construção" de um programa, a serem montados pelos programadores [81].

mais produtivos e utilizados da atualidade [69]. O mecanismo de encapsulamento de dados, por exemplo, é oferecido por linguagens funcionais através de tipos abstratos de dados (ADT ou *abstract data types*), enquanto o polimorfismo é proporcionado por "classes de tipo" (*class types*). Mais informações sobre esses conceitos específicos do paradigma de programação funcional, cujo detalhamento não faz parte do escopo deste documento, podem ser encontradas no tutorial *A Gentle Introduction to Haskell, Version 98* [29].

Em resumo, as principais vantagens oferecidas por linguagens funcionais podem ser listadas em:

- **Maior concisão:** códigos feitos em linguagens funcionais são de 2 a 10 vezes menores do que códigos feitos em linguagens imperativas [38];
- **Maior elegância e facilidade de compreensão:** o *gap semântico* entre o programador e a linguagem é menor no caso de linguagens funcionais. Por exemplo, o conceito de *list comprehension* ("compreensão de listas") facilita extremamente a definição de operações de mapeamento e filtragem de coleções [10];
- **Ausência de *core dumps*:** linguagens funcionais são "fortemente tipadas" e não possuem acesso explícito à memória, sendo impossível tratar um inteiro como um ponteiro, por exemplo, ou haver referências nulas;
- **Reuso de código:** através da utilização de polimorfismo, é possível obter um nível razoável de reuso de código em linguagens funcionais;
- ***Lazy evaluation*:** as expressões são avaliadas sob demanda, isto é, nenhuma parte desnecessária de código é avaliada durante a computação de um resultado;
- ***Literate Programming*:** esta técnica, que combina uma linguagem de programação com uma linguagem de documentação, permite a extração não apenas do código executável como também de sua documentação, a partir de um mesmo código-fonte [12];
- **Alta abstração e modularização:** o conceito de "funções de alta-ordem", por exemplo, permite que funções sejam passadas como argumentos para outras funções, retornadas como resultado de outras funções ou armazenadas como parte de uma estrutura de dados. Além disso, é permitida ao programador a definição de seus próprios operadores infixos;
- **Gerenciamento automático de memória:** o mecanismo de coleta de lixo (*garbage collection*) evita que programadores se preocupem em desalocar explicitamente a memória.

Apesar da série de vantagens apresentadas acima, é reconhecido que a utilização e aplicação atual de linguagens funcionais são limitadas, por uma série de restrições [80], ocasionando uma sub-utilização do potencial dessas linguagens. Algumas dessas deficiências são discutidas a seguir, mais especificamente no contexto da linguagem Haskell.

O primeiro ponto negativo atribuído à utilização de Haskell no desenvolvimento de aplicações diz respeito à performance. É fato que programas desenvolvidos através desta linguagem tendem a ter pior performance quando comparados a programas similares desenvolvidos através de outras linguagens [5]. Contudo, este problema não constitui um em-

pecilho real para aplicações em que a performance não é absolutamente crítica. Wadler [80], em um artigo que, inclusive, defende o uso de C/C++ e questiona o uso de linguagens funcionais, cita que a questão da performance definitivamente não pode ser usada como argumento para justificar a não-utilização de linguagens funcionais. Comprovando tal afirmativa, um trabalho publicado pelo próprio aluno deste Trabalho de Graduação [23], juntamente com seu orientador, constatou que as performances de duas versões de um mesmo jogo não muito complexo, uma desenvolvida em Haskell e outra na linguagem C++, tendem a se igualar quando uma configuração razoável de hardware está presente.

Outras deficiências de linguagens funcionais, entretanto, realmente contribuem para restringir sua aplicação em mais áreas da computação. O conjunto de bibliotecas para adicionar recursos a linguagens funcionais, por exemplo, ainda é limitado. O processo de instalação e manutenção de compiladores, depuradores e outras ferramentas que dão suporte a este tipo de linguagem, por sua vez, não é trivial. Tal afirmação é justificada, essencialmente, devido ao fato de que muitas dessas ferramentas e suas opções só podem ser implantadas através da compilação de seus códigos-fonte e realização de configurações pontuais na máquina do usuário. Como boa parte dos desenvolvedores de ferramentas Haskell são adeptos das plataformas Unix, a questão da configuração correta da máquina de um usuário Windows, de modo a suportar essas ferramentas, exige um esforço ainda maior.

Outro problema ocorre devido ao passado isolacionista das linguagens funcionais, isto é, sua compatibilidade com outras tecnologias é de certa forma limitada. Atualmente, esforços estão sendo realizados com o objetivo de superar tal deficiência, como a integração de Haskell à tecnologia COM, por exemplo [70].

Adicionalmente, a própria falta de popularidade das linguagens funcionais consiste em um problema, visto que elas são menos difundidas do que linguagens como C, C++, C# e Java. Aliando isto ao fato de que a sintaxe de linguagens funcionais difere bastante do padrão apresentado por estas outras linguagens, há um impacto negativo na curva de aprendizado de programadores que desejem migrar para o paradigma de programação funcional.

Por fim, para aplicações cuja performance é crítica, outros tipos de linguagens são mais adequados do que linguagens funcionais. Por exemplo, a construção de uma aplicação de controle de tráfego aéreo, em tempo real, para uma região de movimentação muito intensa, pode abrir mão da produtividade de linguagens funcionais em troca da garantia de um tempo mínimo de execução para suas rotinas.

De todas as deficiências relativas a Haskell e linguagens funcionais em geral, como as apresentadas acima, existe uma bastante crítica, cujo impacto negativo é prontamente reconhecido: a ausência de ambientes integrados de desenvolvimento que dêem suporte ao programador na execução produtiva de suas atividades. Em outras palavras, a etapa de co-

dificação de um projeto de software que utiliza Haskell como linguagem não é beneficiado por mecanismos destinados à melhoria da usabilidade das interfaces manuseadas pelo programador, sendo o processo atual realizado de maneira *ad hoc* e, portanto, mais sujeito a erros[8].

Acreditando na linguagem Haskell como uma alternativa concreta para a implementação de aplicações em larga escala, é exatamente através da resolução dessa última deficiência que este projeto se propõe a contribuir para a comunidade Haskell. A seção a seguir justifica a necessidade por Ambientes Integrados de Desenvolvimento, culminando com o surgimento do Visual Studio .NET 2003.

2.3 Do Vi ao Visual Studio.NET 2003

Na mesma época do surgimento das linguagens de baixo e médio nível, foi consolidada a maneira padrão utilizada até hoje por desenvolvedores para programar uma aplicação: a escrita de código-fonte, que substituiu o uso de cartões perfurados e fios elétricos. Nesses estágios iniciais, portanto, a etapa de implementação de um processo de desenvolvimento de software envolvia ferramentas independentes, utilizadas ciclicamente: um editor de texto simples (como o Vi[9]), o compilador da linguagem utilizada e um mecanismo de depuração trivial, quando não a inserção de comandos de impressão de mensagens no próprio código-fonte [7].

Com o tempo, os programadores passaram a utilizar editores de texto direcionados à sintaxe [15,75], de modo a obter uma melhor visualização de seu código-fonte através de recursos como colorização diferenciada para cada tipo de *token* da linguagem. Até hoje, editores direcionados à sintaxe existem e são bastante úteis, principalmente por serem leves e simples (ao contrário das atuais IDEs), como é o caso do UltraEdit-32 [40]. Nessa ferramenta, para oferecer um suporte mínimo à edição de uma linguagem, basta adicionar, em um arquivo texto de configuração do editor, uma entrada que associa a linguagem a extensões de arquivos, *tokens* e cores.

---

[8] Recentemente, surgiram alguns projetos para a criação/extensão de IDEs para Haskell. Entretanto, esses projetos ainda não foram difundidos na prática, devido ao próprio fato de serem recentes ou por causa de suas limitações, como mostra a seção “Trabalhos Similares e Anteriores”.

[9] O Vi é considerado por alguns como o primeiro editor de texto da história [48]. Entretanto, o título é também disputado por outros editores, como o Emacs [42].

À título de ilustração, a Figura 5 apresenta a edição de um arquivo, contendo código-fonte da linguagem C++, em um editor simples, como o Microsoft Notepad. Uma comparação entre essa figura e a Figura 6, que exibe a edição do mesmo arquivo em um editor direcionado à sintaxe, revela como é mais interessante trabalhar com a segunda opção.

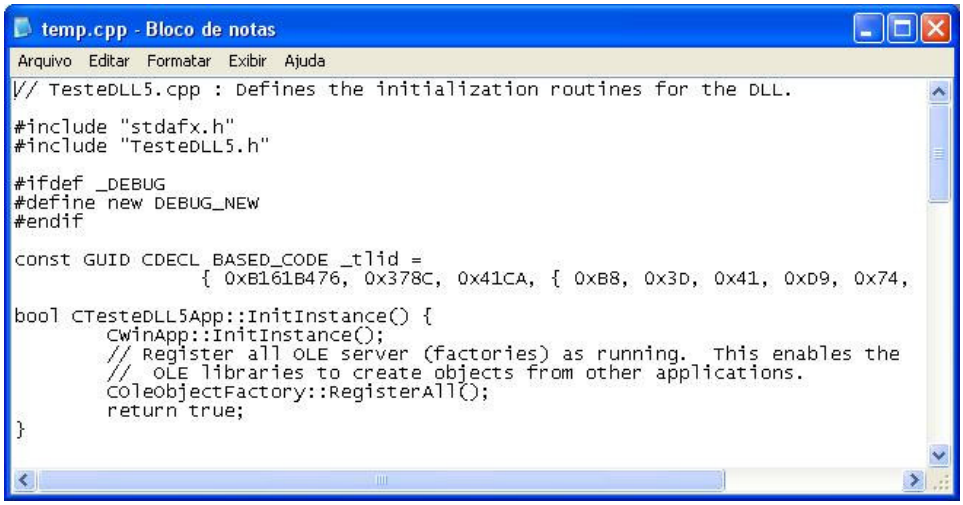

Figura 5 – Edição de código-fonte em um editor simples, como o Notepad

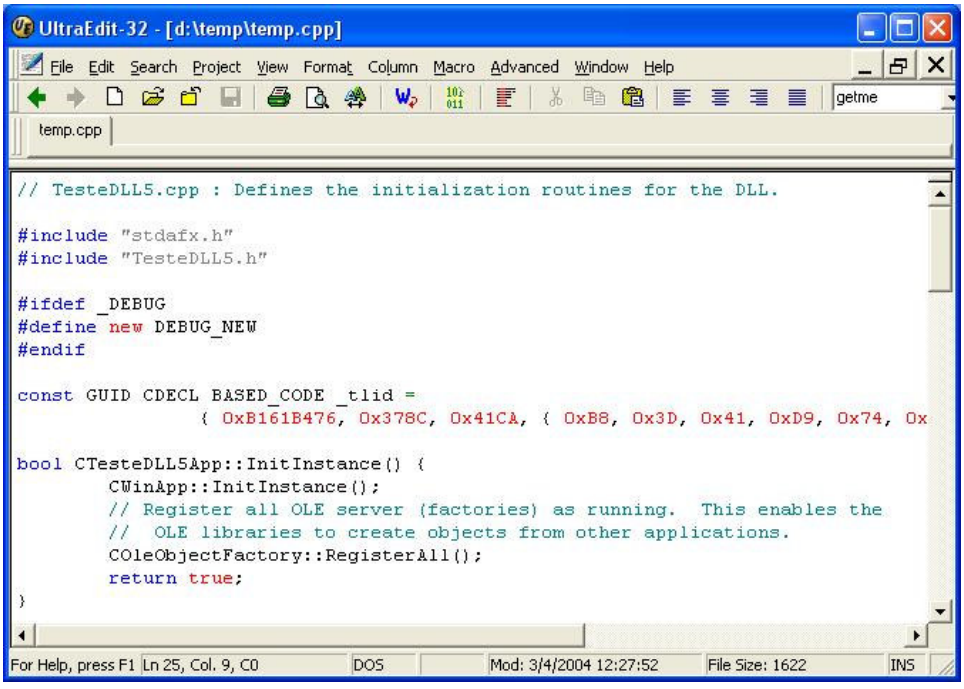

Figura 6 – Edição do mesmo código-fonte, agora em um editor direcionado à sintaxe

Apesar de interessantes, logo foi percebido que apenas editores direcionados à sintaxe não eram suficientes. As etapas de compilação e depuração ainda eram feitas à parte, implicando no uso de diferentes ferramentas e impactando negativamente na produtividade do desenvolvedor. Quando essas ferramentas foram integradas em apenas uma, nasceram, portanto, as primeiras IDEs, inicialmente oferecendo apenas as três funcionalidades básicas do ciclo de implementação de uma aplicação: edição, compilação e depuração [7]. Como exemplo, pode-se citar um IDE pioneira, criada em 1964, para a linguagem Dartmouth Basic [1], que originaria a linguagem BASIC alguns anos depois.

Com o tempo, atendendo às crescentes necessidades dos desenvolvedores, as IDEs passaram a oferecer conjuntos muito mais ricos de funcionalidades, como o suporte a diferentes tipos de arquivos (não só texto como também gráficos), analisadores estatísticos, compiladores incrementais, "navegadores" de documentos e a manutenção da dependência entre os mesmos [17]. Acompanhando o surgimento de diversas tecnologias da computação, as necessidades dos desenvolvedores aumentaram ainda mais, demandando, conseqüentemente, uma evolução contínua das IDEs e do conjunto de funcionalidades oferecido pelas mesmas. Se, antigamente, o modelo de desenvolvimento era bastante simples (códigos-fonte escritos em uma única linguagem com o objetivo de rodar em apenas uma plataforma específica), hoje em dia o cenário é extremamente diferente. Alguns exemplos das necessidades atuais de desenvolvedores de software são [7]:

- Dominar um conjunto diversificado de **linguagens de programação básicas** (como um subconjunto qualquer de C++, Java, C#, Visual Basic, Delphi, Haskell, etc.);
- Conhecer as peculiaridades das diferentes **plataformas** em que suas aplicações serão implantadas (Microsoft Windows, Linux, Pocket PC, Palm, etc.);
- Entender as **linguagens de descrição de dados** para a Web (HTML, WML, XML, XSLT, XSD, DTD, CSS);
- Saber lidar com **tecnologias de acesso a bases de dados** (SQL, JDBC, RDO, ADO, ADO.NET, etc.);
- Compreender a aplicação de **tecnologias de *scripting*** às camadas de frente da aplicação (JavaScript, VBScript, ASP, ASP.NET, Servlets, JSP, CGI, PHP, etc.);
- Entender as **tecnologias de componentização** de aplicações (EJB, COM, COM+, etc.);
- Estar atualizado em relação a **conceitos recentes**, como os Web Services, que deram uma nova dinâmica à utilização da *arquitetura orientada a serviço* [6].

Em resumo, a produtividade do programador atual está diretamente relacionada à capacidade de integração de tecnologias oferecida pelas ferramentas que ele utiliza. Sob o *slogan* "visionário, porém prático" [62], o Visual Studio .NET 2003 é uma IDE, desenvolvida pela Microsoft, cujo objetivo principal é atender à necessidade pela produtividade, através da automação de tarefas e integração de tecnologias [60].

O VS.NET é parte principal da estratégia da Microsoft para a difusão do .NET Framework, a recente plataforma para desenvolvimento de software criada pela empresa. O detalhamento do .NET Framework foge ao escopo deste trabalho; mais informações sobre a tecnologia podem ser encontradas no *Microsoft .NET Framework Developing Center* [56].

O Visual Studio .NET 2003 retrata adequadamente o estado-da-arte dos ambientes integrados de desenvolvimento. Entre algumas de suas funcionalidades, muitas delas exclusivas, pode-se destacar:

- Suporte ao desenvolvimento e comunicação de **projetos em diferentes linguagens** de programação;
- **Desenvolvimento visual de formulários** Windows, Web ou para dispositivos móveis, inclusive através do conceito de herança entre formulários;
- **Rico suporte à edição**, através das funcionalidades oferecidas pelo *Microsoft Intellisense®*;
- **Ajuda dinâmica**, sensível a contexto;
- Amplo conjunto de ***wizards*** para facilitar a execução de tarefas, como a criação de projetos e classes, por exemplo;
- **Depuração** com um conjunto rico de funcionalidades (depuração remota, depuração multiliguagem, *breakpoints*, *quick-watches*, etc.);
- Criação de **projetos instaladores**, baseados na tecnologia *Microsoft Installer*, para a implantação de aplicações;
- Suporte ao desenvolvimento, depuração e implantação de **Web Services**;
- Desenvolvimento de **páginas para a Web** (como HTML, ASP e ASP.NET) sem a necessidade da utilização de scripts;
- RAD (***rapid application development***) também para servidores, integrando ferramentas de administração como *logs* de eventos, bancos de dados e filas de mensagens;
- Edição e validação de **arquivos XML**;
- Integração com **ferramentas de controle de versão**;
- Integração com **ferramentas de teste** de performance, funcionalidade e carga, inclusive para a Web.

Existem, atualmente, quatro diferentes versões do Visual Studio .NET 2003, diferenciadas pelo pacote de softwares que acompanha a IDE e por funcionalidades para o desenvolvimento de aplicações servidoras:

- Visual Studio .NET 2003 Enterprise Architect;
- Visual Studio .NET 2003 Enterprise Developer;
- Visual Studio .NET 2003 Professional;
- Visual Studio .NET 2003 Academic;

A versão *Academic* do VS.NET tem como foco a facilitação de atividades executadas por professores e alunos. Isso é possível através da ferramenta Assignment Manager, que acompanha esta versão do VS.NET e é apresentada na seção seguir.

2.4 A Ferramenta Assignment Manager

O Assignment Manager (AM) é uma ferramenta acadêmica, que acompanha o Visual Studio .NET 2003 Academic, cuja principal finalidade é gerenciar, para professores e alunos, o processo de publicação e submissão de tarefas acadêmicas que envolvem codificação. Esta ferramenta, na verdade, é composta por três componentes: *Assignment Manager Server*, *Assignment Manager Faculty Client* e *Assignment Manager Student Client*. O relacionamento entre esses três componentes pode ser visualizado na Figura 7.

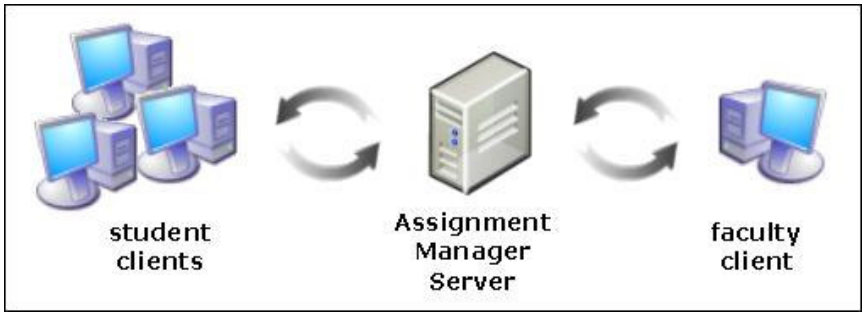

Figura 7 – Relacionamento entre os componentes da ferramenta Assignment Manager

O *Assignment Manager Server*, que atende a solicitações dos outros dois componentes, deve ser instalado em uma máquina contendo o servidor Web da Microsoft (*Internet Information Services* ou IIS [55]), o serviço de enfileiramento de mensagens MSMQ (*Microsoft Message Queuing*) [57] e o *desktop engine* do SQL Server 2000 [59] (que acompanha o próprio SQL Server mas também pode ser instalado à parte, gratuitamente). Por se tratar de uma aplicação servidora, o *AM Server* não possui interface gráfica, devendo ser administrado a partir de ferramentas de administração do próprio Windows, como o *log* de eventos.

Estudantes devem ter instalado em suas máquinas o *AM Student Client* e, o professor, o *AM Faculty Client*. Ambas as ferramentas estão embutidas no próprio Visual Studio .NET 2003 Academic, sendo suas funcionalidades acessadas a partir do menu da IDE, que aciona o carregamento de interfaces (em HTML) dentro do próprio ambiente. Através do *Faculty Client*, um professor pode criar um novo curso, publicar exercícios e atribuir notas. Utilizando o *Student Client*, por sua vez, estudantes podem acessar os cursos e exercícios disponíveis, submeter respostas dos exercícios e acessar as notas atribuídas pelo professor. A Figura 8 mostra um *screenshot* do AM em execução.

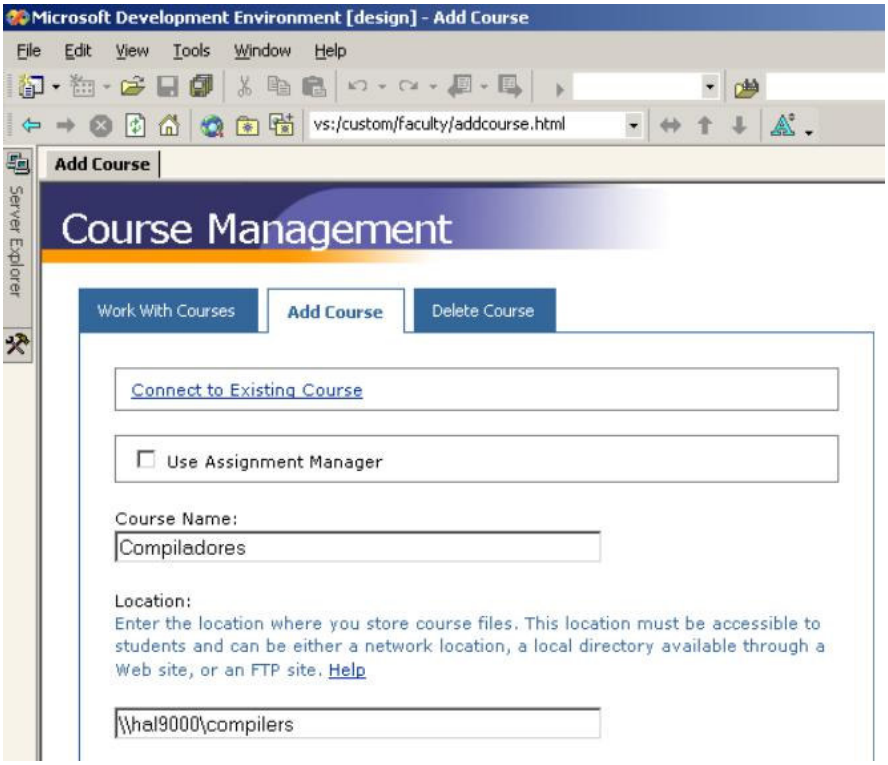

Figura 8 – Assignment Manager em execução (*Facuty Client*)

Para entender o conceito de "exercício" no Assignment Manager, é necessário entender o conceito de "solução" e "projeto" no VS.NET. Grosso modo, a IDE define um projeto como um conjunto de arquivos (código-fonte na linguagem escolhida, páginas HTML, imagens, ícones, etc.) que podem ser compilados para a construção um *output* específico. Esse *output* pode ser uma aplicação Windows, uma aplicação de console, uma dll ou um Web Service, por exemplo. Uma solução, por sua vez, nada mais é do que uma coleção de projetos, que gerencia as interdependências entre eles e realiza operações comuns a todos os projetos, como controle de versão, por exemplo. Um exercício do Assignment Manager, portanto, nada mais é do que um projeto criado pelo professor no VS.NET que contém, possivelmente, trechos de código omitidos, a serem preenchidos pelos estudantes.

Para publicar um exercício através do *Faculty Client*, o professor precisa, inicialmente, estar com um projeto aberto no VS.NET (e conseqüentemente com uma solução aberta também). Nos arquivos contendo o código-fonte do projeto, o professor pode marcar trechos de código e solicitar ao *AM Faculty Client* que considere o trecho selecionado como "*student code*", isto é, código que deverá ser elaborado pelos estudantes para a conclusão do exercício. Após a solicitação da publicação do exercício pelo professor através do menu do VS.NET, o *AM Server* se encarrega de omitir os trechos de código marcados como "*student code*". Adicionalmente, o professor pode especificar ao AM Server a execução de *auto-build*

e/ou *auto-check* nas respostas submetidas pelos alunos. O *auto-build* verifica que o projeto está compilando sem erros, enquanto o *auto-check* verifica a corretude da resposta do aluno através da especificação de arquivos de entrada e de saída para a aplicação. Caso a saída especificada pelo arquivo de saída seja igual à obtida através da execução do projeto-resposta do aluno, a partir do arquivo de entrada, o teste é considerado bem-sucedido.

Quando os estudantes fizerem o *download* do exercício a partir do *AM Student Client*, no lugar dos trechos de código marcados como *student code* pelo professor agora aparecerão comentários, indicando que houve uma supressão de código nestes locais. Uma vez concluídos os exercícios, os estudantes devem submetê-los ao *AM Server*, através do *AM Student Client*. Após a execução dos testes de *auto-build*/*auto-check* pelo *AM Server*, o professor pode verificar o projeto-resposta do aluno para, por fim, atribuir sua nota final, acessível pelos alunos posteriormente.

Como pôde ser observado, os conceitos por trás da ferramenta Assignment Manager são simples, embora bastante aplicáveis na prática. De modo a enriquecer o conjunto de funcionalidades oferecido pela ferramenta, existem, atualmente, diversos projetos para estendê-la, além do projeto VHS-AM. Esses e demais projetos relacionados a este Trabalho de Graduação serão apresentados na seção a seguir.

2.5 Trabalhos Similares e Anteriores

2.5.1 Projetos de Extensão do Assignment Manager

Os projetos apresentados abaixo foram originados a partir da mesma RFP que viabilizou o projeto VHS-AM (mais detalhes no Capítulo 1). O *status* atual de cada projeto pode ser encontrado a partir da área de projetos da *Microsoft Academic Shared Source Community* (ASSC) [61].

2.5.1.1 Monitoração do desempenho do aluno através do Assignment Manager

A Universidade de Monash [67], na Austrália, pretende estender o AM de modo a permitir que o conteúdo de um projeto em desenvolvimento por um estudante, assim como o progresso e erros advindos de sua compilação, sejam capturados em diversos momentos, permitindo a criação de um relatório para que o professor entenda melhor a evolução do aluno na realização do exercício[10].

[10] Um estudante de mestrado ITA (Instituto Tecnológio da Aeronáutica), Daniel Pegas, também está desenvolvendo um trabalho similar, ainda que em fase inicial. Atualmente, algumas experiências estão sendo trocadas entre ele e a equipe do projeto VHS-AM.

2.5.1.2 Extensão do conceito de "exercícios" do Assignment Manager

A Universidade de Hull [77], na Inglaterra, está implementando funcionalidades para ampliar o campo de aplicação do AM na comunidade acadêmica, ao permitir a submissão de documentos e exercícios que não sejam apenas projetos do VS.NET. Além disso, o projeto também pretende integrar o AM com ferramentas desenvolvidas na própria universidade, para obtenção de *feedback* de trabalhos práticos submetidos aos alunos por um professor.

2.5.1.3 Extensão do Assignment Manager para cursos de programação introdutórios

A Universidade de Yale [82], nos EUA, está estendendo o AM para que o mesmo se adapte a cursos de programação introdutórios, suportando a atribuição de notas a exercícios baseados na programação da GUI (*graphical user interface*).

2.5.1.4 Melhorias Gerais do Assignment Manager

A Universidade Estadual Paulista Julio de Mesquita Filho (UNESP) [79] está implementando um conjunto de melhorias para o AM, como a divisão de estudantes em classes dentro de um mesmo curso, o acesso a informações de estudantes por parte de professores e administradores do *AM Server*, extração de análises estatísticas e acesso a informações mais ricas sobre exercícios pendentes e submetidos.

2.5.2 Ferramentas de Suporte a Projetos e Edição de Código Haskell

Esta subseção apresenta algumas opções atualmente disponíveis ao programador Haskell para o desenvolvimento de aplicações. Existem alguns esforços relativos à extensão de outras IDEs, além do VS.NET, para suportar a utilização de Haskell. Outros projetos, por sua vez, apresentam ambientes desenvolvidos exclusivamente para Haskell.

2.5.2.1 Extensão do Eclipse

O Eclipse [16] é uma IDE aberta e extensível, implementada em Java e que roda em várias plataformas. Uma expressão que define muito bem a ferramenta é "uma IDE para tudo, e nada em particular" [74]. Alguns projetos realizados com base no Eclipse tornaram-se muito bem-sucedidos, como a extensão da IDE para Java. Recentemente, Leif Frenzel publicou a versão 0.1.0 de um projeto que estende o Eclipse para o suporte a Haskell [22]. No momento, este projeto suporta apenas a compilação, execução e edição de código Haskell, com recursos como *syntax coloring*[11] e indicação de mensagens de erro em uma barra de ferra-

[11] O detalhamento dos recursos oferecidos por uma IDE em relação ao suporte à edição será apresentado no Capítulo 3.

mentas. A compilação é feita invocando-se o GHC. Um *screenshot* da ferramenta em execução pode ser visualizado na Figura 9.

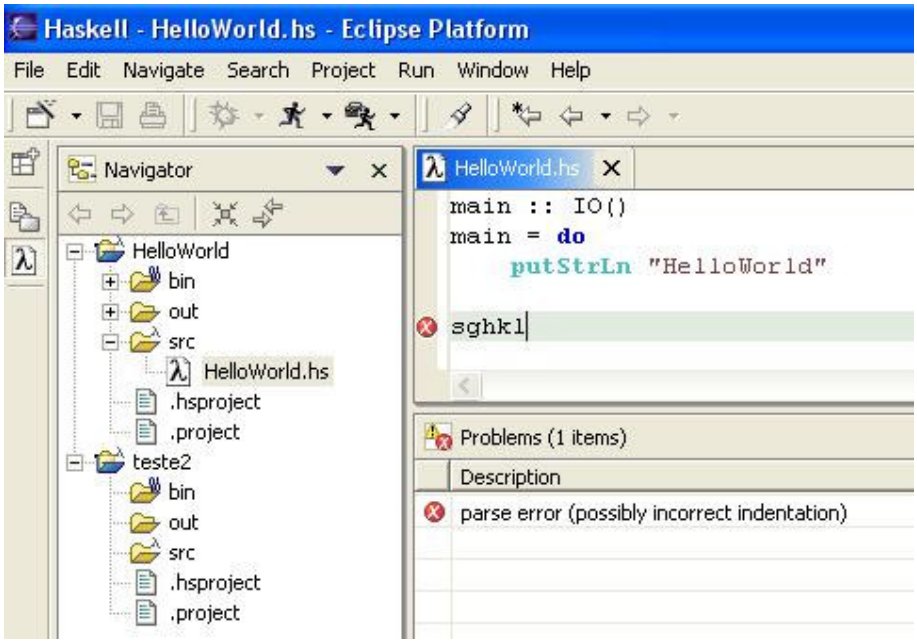

Figura 9 – A IDE Eclipse com suporte a Haskell

Apesar desta ser uma interessante abordagem, o autor deste trabalho acredita que o modelo de extensão do VS.NET, apresentado no Capítulo 4, é consideravelmente mais completo do que o do Eclipse, podendo oferecer um suporte a Haskell mais satisfatório.

2.5.2.2 WinHugs

O WinHugs é uma versão visual e mais amigável, para Windows, da ferramenta Hugs [35], um interpretador para a linguagem Haskell. Não é oferecido nenhum suporte para a edição de código; o WinHugs apenas dispara o editor definido pelo programador. A principal vantagem da utilização do WinHugs consiste na sua interface, que permite uma utilização mais amigável dos comandos do Hugs original (carregar um módulo haskell, executar a função main, sair da aplicação, copiar, colar, recortar, etc.) e também navegar pelas estruturas de dados das bibliotecas-padrão de Haskell. Um *screenshot* da ferramenta em execução pode ser visualizado na Figura 10.

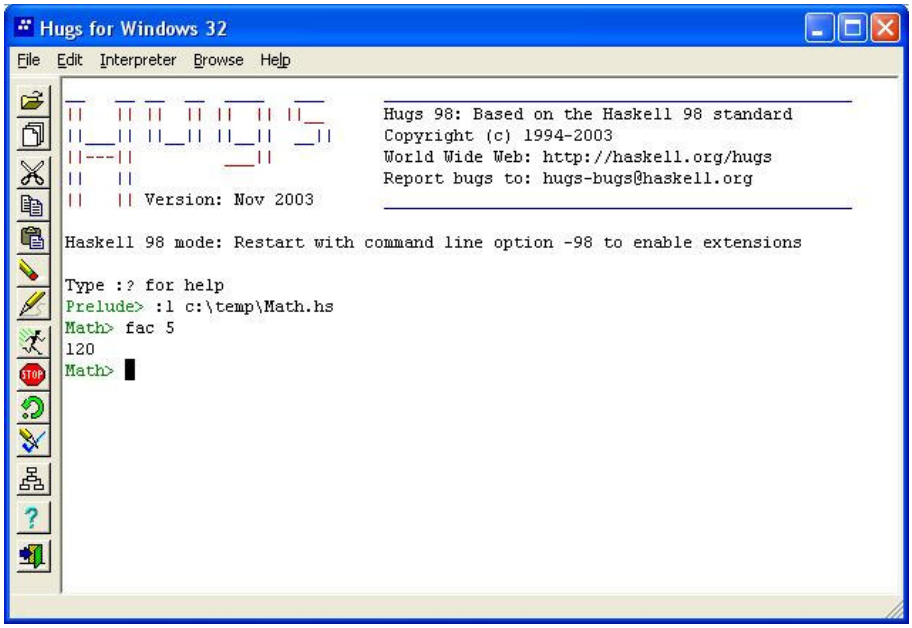

Figura 10 – WinHugs em execução

Uma desvantagem evidente do WinHugs, além de suas limitações funcionais, é o fato dele apenas interpretar (e não compilar) código Haskell, diminuindo seu campo de aplicação. Entretanto, a equipe do projeto VHS-AM reconhece que a interpretação de código é importante em determinados casos (como a análise estática de um módulo Haskell em desenvolvimento) e pretende incluir no VS.NET estendido uma barra de ferramentas que permita ao usuário a invocação de comandos de um interpretador, como o próprio Hugs ou o GHCi (que acompanha o GHC).

2.5.2.3 hIDE

O hIDE é uma IDE para Haskell que, semelhantemente ao WinHugs, não possui suporte nativo à edição de código, mas se comunica com editores externos como o Vim ou Emacs. Ela permite a criação de projetos Haskell e suporta o processo de compilação, invocando o GHC. Adicionalmente, é possível navegar pelos arquivos e módulos Haskell do projeto, além de obter relatórios contendo uma documentação simples de cada módulo e gerenciar uma lista das tarefas a fazer ("*todo list*"). Um *screenshot* da ferramenta em execução pode ser visualizado na Figura 11.

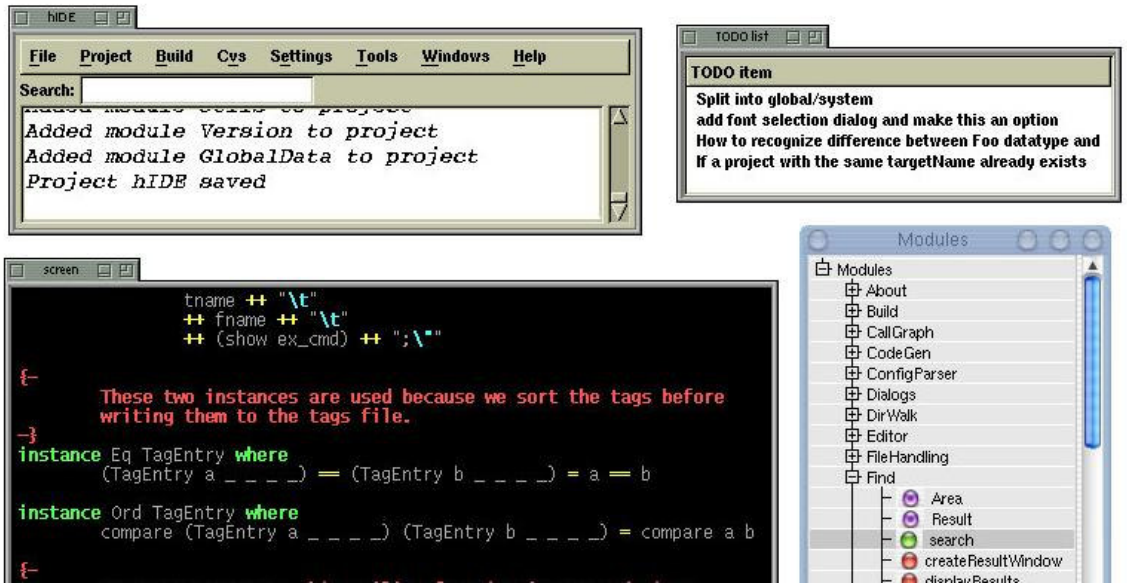

Figura 11 – Ferramenta hIDE em execução

O hIDE está disponível apenas para plataformas Unix, o que diminui consideravelmente seu campo de aplicação. Adicionalmente, o fato desta IDE ser feita do zero implica em muito trabalho para a adição de novas funcionalidades, que poderiam ser mais facilmente obtidas a partir da adaptação de uma IDE extensível, como o Eclipse e o VS.NET.

2.5.2.4 JCreator

O JCreator [27] é uma IDE extensível, feito em Java, para a plataforma Windows. Neste projeto, a IDE foi estendida para suportar a criação de projetos Haskell, edição com *syntax coloring*, visualização de mensagens de erro em uma barra de ferramentas e invocação do interpretador Hugs. Um *screenshot* da ferramenta em execução pode ser visualizado na Figura 12.

Devido à invocação do Hugs, e não do GHC, o suporte a Haskell oferecido pela extensão do JCreator não suporta o processo de compilação. Adicionalmente, assim como no do suporte a Haskell através do Eclipse, o JCreator parece oferecer um conjunto de funcionalidades limitado quando comparado ao VS.NET e seu rico modelo de extensão.

2.5.2.5 Extensão do KDevelop

O KDevelop [44] é uma IDE que surgiu em 1998 para suportar o KDE [43], ambiente de *desktop* gráfico para Linux e Unix. Desde então, ele é utilizado para suportar também diversas linguagens de programação. Atualmente, a extensão para Haskell oferece apenas *templates* de projeto, a ligação com o compilador GHC e inserção de tópicos de documentação no código-fonte.

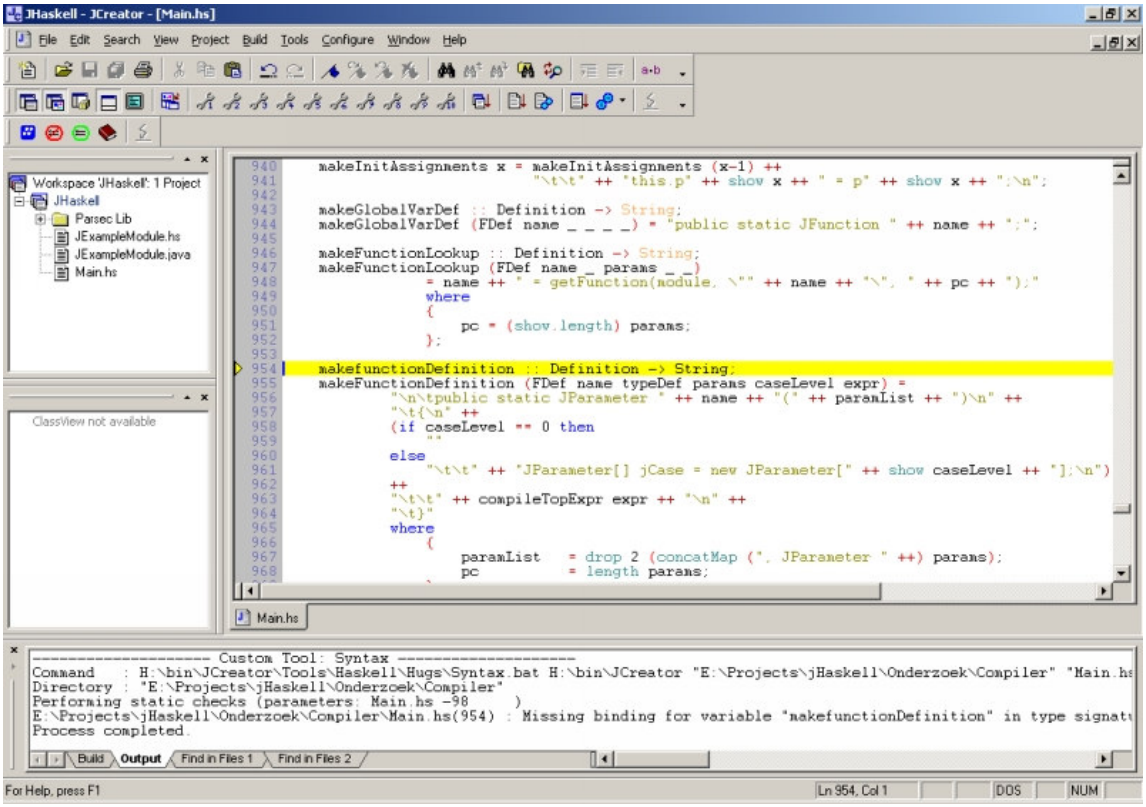

Figura 12 – JCreator com suporte a Haskell

2.5.3 Projetos de Suporte a Outras Linguagens no VS.NET

Uma das principais vantagens da utilização do .NET Framework consiste na grande variedade de linguagens disponíveis para a escolha do programador. Atualmente, muitas linguagens foram ou estão sendo portadas para uma versão .NET, do Cobol .NET [9] ao Visual Basic .NET. Entretanto, além de portar a linguagem, é necessário também prover um suporte à utilização da mesma para garantir a satisfação de seus programadores. No caso dessas novas “linguagens .NET”, este suporte é oferecido, na maioria das vezes, através da extensão do próprio VS.NET. Os criadores do projeto brasileiro Lua.NET [41], por exemplo, consideram a integração com o VS.NET uma etapa futura essencial a ser realizada no projeto.

A linguagem SML.NET, atualmente suportada pelo VS.NET [78], merece uma atenção especial. Além dessa ser uma linguagem funcional, como Haskell, a abordagem utilizada pelo SML.NET para a extensão do VS.NET é bem semelhante à escolhida pelo projeto VHS. É interessante observar que, assim como as novas linguagens .NET, há também o interesse em oferecer suporte do VS.NET a linguagens mais antigas, de modo a enriquecer a experiência do programador em sua utilização. Este é o caso da linguagem Python [3], de Perl [2] e da própria linguagem Haskell.

3. ESCOPO

Este capítulo apresenta o escopo do projeto VHS-AM através de seus casos de uso e requisitos não-funcionais (RNFs), com dois objetivos. O primeiro é permitir que o leitor visualize com mais clareza o produto final a ser atingido pelo projeto VHS-AM. O segundo é viabilizar que o modelo de extensão do VS.NET, detalhado no capítulo a seguir, seja discutido em termos dos casos de uso e RNFs aqui apresentados, justificando de maneira mais precisa a escolha da abordagem utilizada no projeto.

De acordo com a metodologia Pro.NET, apresentada em mais detalhes no Capítulo 6, o aluno deste Trabalho de Graduação, por executar funções de gerência de projeto, foi o responsável pela elaboração do Documento de Especificação Funcional (DEF), que detalha e relaciona os casos de uso e requisitos não-funcionais da aplicação a ser desenvolvida. Em uma etapa posterior, este documento foi validado com toda a equipe, permitindo o início da Fase de Desenvolvimento do projeto. As informações contidas neste capítulo, portanto, têm como base o DEF já validado pela equipe do projeto VHS-AM.

3.1 Atores da Aplicação

A metodologia Pro.Net define que um passo anterior à identificação dos casos de uso de uma aplicação consiste na definição de seus atores, isto é, dos papéis desempenhados pelos usuários da aplicação. Além de representar pessoas, os atores também podem ser dispositivos de hardware ou até mesmo outras aplicações que devam trocar informações com a aplicação a ser desenvolvida. A lista dos atores identificados no projeto é exibida na Tabela 1, que mostra também uma breve descrição para cada um deles.

Tabela 1 – Atores identificados no projeto VHS-AM

Ator	Descrição
Programador Haskell	Indivíduo que utilizará o VS.NET estendido com o objetivo de elaborar aplicações em Haskell.
Professor Assignment Manager	Indivíduo que utilizará o *AM Faculty Client* (beneficiado pelo VS.NET estendido) com o objetivo de criar e publicar exercícios acadêmicos da linguagem Haskell que envolvam codificação. Este ator é uma especialização do ator Programador Haskell.
Aluno Assignment Manager	Indivíduo que utilizará o *AM Student Client* (beneficiado pelo VS.NET estendido) com o objetivo de responder e submeter exercícios acadêmicos da linguagem Haskell que envolvam codificação. Este ator é uma especialização do ator Programador Haskell.
GHC (Glasgow Haskell Compiler)	Compilador Haskell a ser eventualmente invocado pelo VS.NET.

3.2 Convenções

Os casos de uso foram rotulados unicamente através de uma numeração específica, que inicia com o identificador [UC01] e prossegue sendo incrementada. O mesmo se aplica para requisitos não-funcionais, que iniciam com [RNF01].

Ao contrário do Documento de Especificação Funcional do projeto, este capítulo irá apresentar, apenas, a descrição, o ator relacionado (em texto sublinhado) e um esboço de interface para cada caso de uso, sendo omitidos atributos como prioridade, casos de uso relacionados, entradas, pré-condições, saídas, pós-condições e fluxos de execução (principal, alternativo e de erro). As informações omitidas podem ser consultadas no próprio DEF do projeto, caso desejado.

3.3 Casos de Uso do Projeto VHS-AM

3.3.1 [UC01] Desenvolver código Haskell com suporte à edição

Ao codificar em Haskell, o Programador Haskell deve obter do VS.NET facilidades visuais que dão suporte ao processo de programação, para aumentar sua produtividade. Este caso de uso, na verdade, foi dividido em 6 "subcasos de uso", apresentados abaixo:

- **[UC01A] Syntax coloring:** diferentes cores são associadas a cada elemento do código, como palavras-chave da linguagem Haskell, comentários e operadores, por exemplo.
- **[UC01B] Brace matching:** ao se fechar ou passar o cursor sobre um parêntese/chave, tanto o parêntese/chave como seu par devem ser ressaltados em negrito, de modo que os limites do bloco de texto envolvido sejam identificados com mais facilidade.
- ***IntelliSense:*** significa um conjunto de funcionalidades sensíveis a contexto, acionadas à medida que o programador dispara alguns eventos específicos. As funcionalidades *IntelliSense* a serem suportadas, em ordem de prioridade, são:
    - **[UC01C]** ***Quick info:*** o tipo de um identificador é exibido quando o ponteiro do mouse "repousa" por algum tempo sobre o mesmo.
    - **[UC01D]** ***Word-complete:*** ao ser pressionado um conjunto de teclas específicas (como CTRL + barra de espaço) ou ao ser selecionado um membro de uma *popup list* (ver abaixo), a "semi-palavra" em cujo final está o cursor é completada.
    - **[UC01E]** ***Method tip:*** ao ser pressionado um conjunto de teclas específicas (como CTRL + SHFT + barra de espaço) logo após o nome de um método, seu tipo de retorno e os tipos de seus parâmetros são exibidos.
    - **[UC01F]** ***Member list popup:*** ao ser pressionado um conjunto de teclas específicas (como CTRL + J) ou ao ser teclado um operador de acesso a membro (como "."), uma lista com todos os identificadores disponíveis é exibida, na forma de *popup*.

Um esboço de interface para este caso de uso pode ser visualizado através da Figura 13

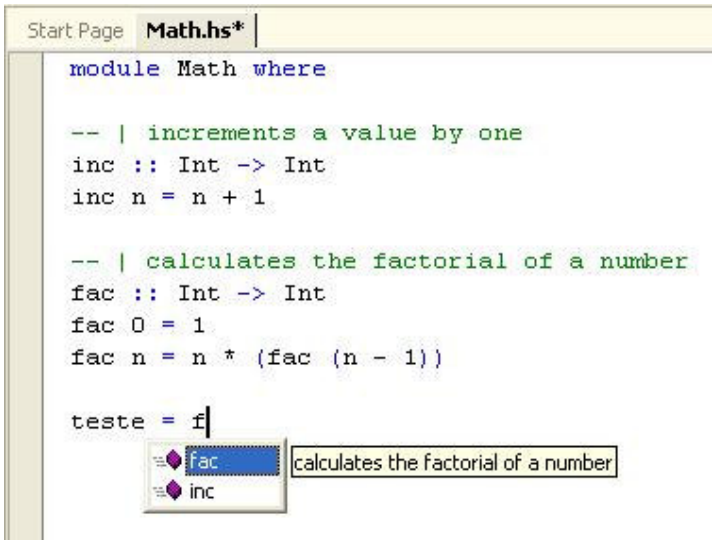

Figura 13 – Esboço de interface para o caso de uso UC01

3.3.2 [UC02] Criar um novo projeto do tipo *Haskell Console Application*

Mais do que simplesmente editar código em Haskell, o Programador Haskell deve ser capaz de agrupar logicamente módulos Haskell em um projeto (que, naturalmente, pertence a uma solução). Isso é necessário não apenas para permitir a compilação e execução do código, como também para a utilização integrada do Assignment Manager com a solução, além de possibilitar que diferentes recursos (como imagens, relatórios, etc.) sejam também associados ao projeto no futuro. Um esboço de interface para este caso de uso pode ser visualizado através da Figura 14.

3.3.3 [UC03] Criar e anunciar exercícios em Haskell através do AM

O Professor Assignment Manager deve ser capaz de utilizar a linguagem Haskell, como qualquer outra linguagem original do VS.NET, para criar e disponibilizar exercícios que envolvam codificação. Estes exercícios podem conter *student code*, a ser suprimido pelo *AM Server* e que deve ser preenchido pelos alunos na resolução do exercício. Um esboço de interface para este caso de uso pode ser visualizado através da Figura 15.

3.3.4 [UC04] Resolver e submeter exercícios em Haskell através do AM

Assim como o professor, o Aluno Assignment Manager também deve ser capaz de trabalhar com a linguagem Haskell do mesmo modo que qualquer outra linguagem original do VS.NET, beneficiando-se do suporte oferecido pela IDE estendida para a resolução dos exercícios publicados. A Figura 16 apresenta um esboço de interface para este caso de uso.

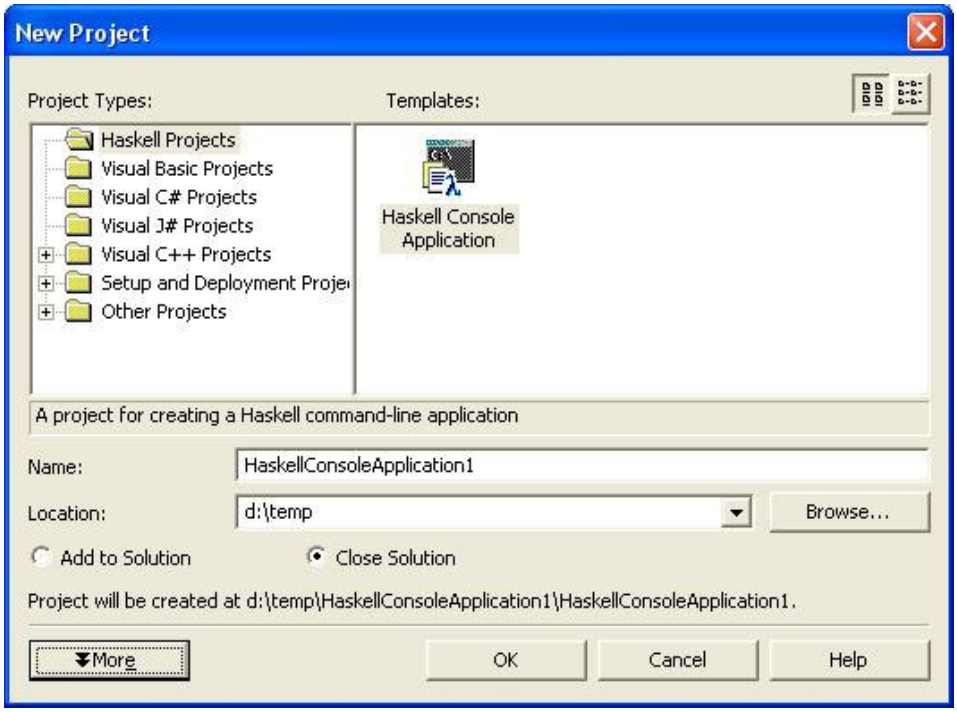

Figura 14 – Esboço de interface para o caso de uso UC02

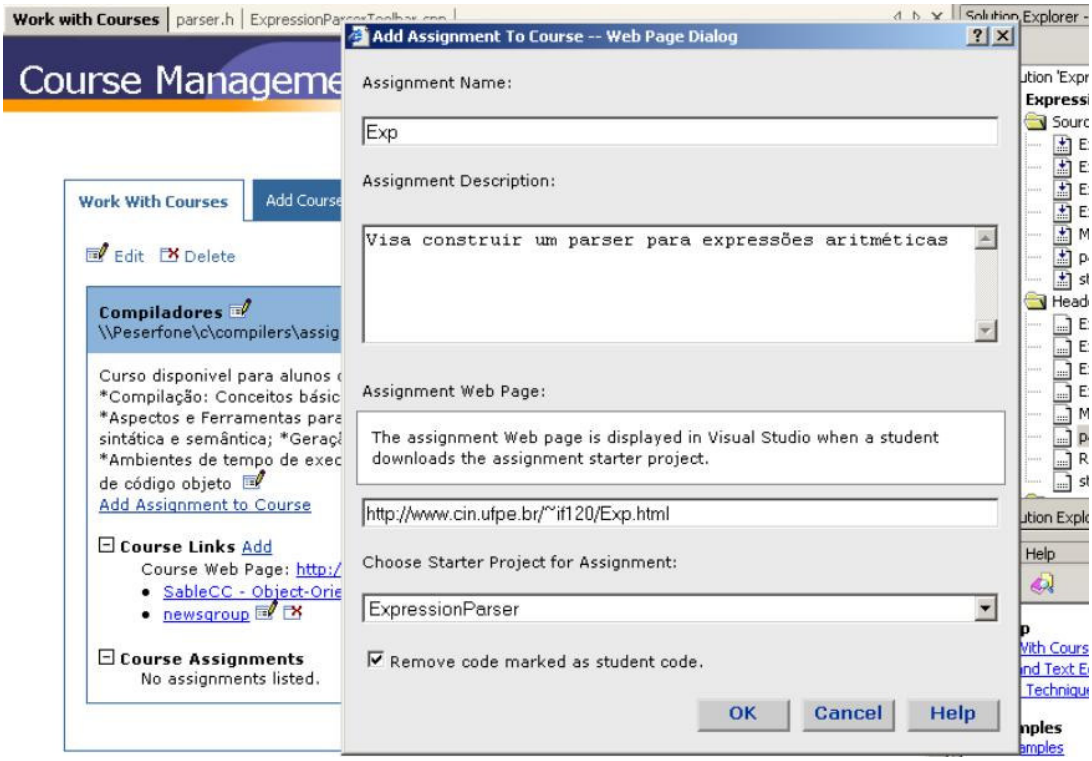

Figura 15 – Esboço de interface para o caso de uso UC03

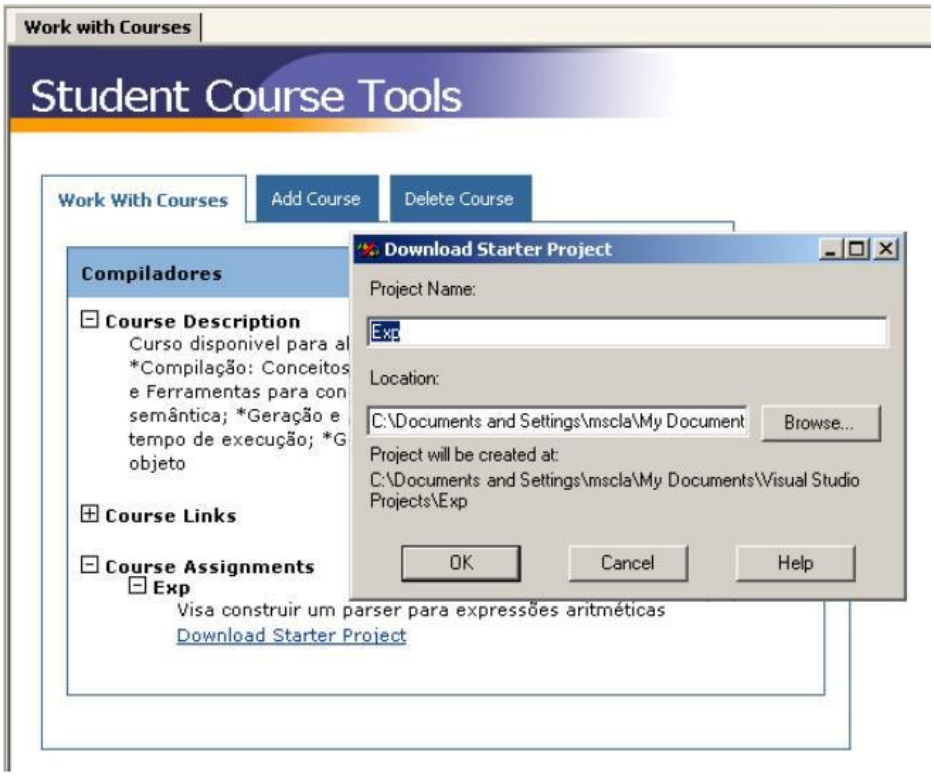

Figura 16 – Esboço de interface para o caso de uso UC04

3.3.5 [UC05] Adicionar um novo módulo Haskell a um projeto Haskell

Um projeto não deve estar limitado por apenas um módulo Haskell. Desse modo, é necessário permitir que o Programador Haskell possa adicionar mais módulos de acordo com suas necessidades. A Figura 17 apresenta um esboço de interface para este caso de uso.

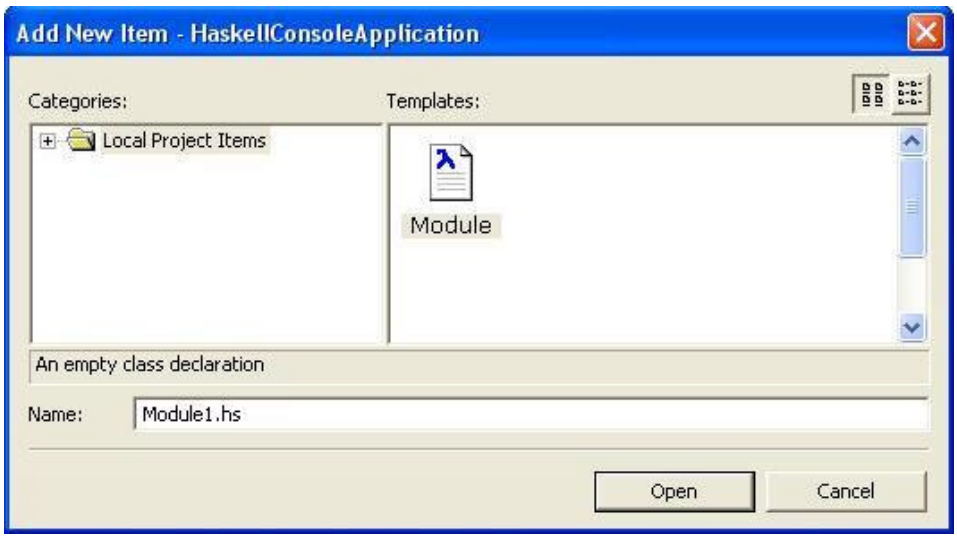

Figura 17 – Esboço de interface para o caso de uso UC05

3.3.6 [UC06] Navegar por um *module viewer* específico para Haskell

De modo a permitir que o Programador Haskell compreenda melhor a estrutura de seu programa, é interessante que um visualizador de módulos, funções, classes e tipos esteja disponível. Um esboço de interface para este caso de uso pode ser visualizado através da Figura 18.

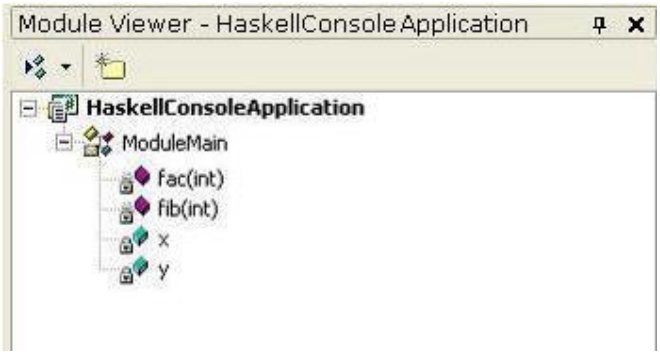

Figura 18 – Esboço de interface para o caso de uso UC06

3.3.7 [UC07] Obter mensagens de erro dinamicamente

Enquanto o Programador Haskell está programando, ele deve ser capaz de saber se existe algum erro em seu código antes mesmo do processo de compilação. Desse modo, trechos contendo erros de código devem aparecer sublinhados em vermelho. Este caso de uso, na verdade, se divide em dois subcasos:

- **[UC07A]** Indicação de erros léxicos/sintáticos
- **[UC07B]** Indicação de erros semânticos.

Um esboço de interface para este caso de uso pode ser visualizado através da Figura 19.

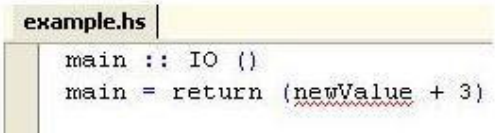

Figura 19 – Esboço de interface para o caso de uso UC07

3.3.8 [UC08] Visualizar mensagens de erro na *taskbar*

O Programador Haskell deve ser capaz de visualizar, rapidamente, todos os erros da aplicação que ele está desenvolvendo, através da *taskbar* do VS.NET. Devem estar presentes na *taskbar* não apenas os erros identificados dinamicamente (UC07) como também novos erros que eventualmente surjam após a compilação da aplicação. Um esboço de interface para este caso de uso pode ser visualizado através da Figura 20.

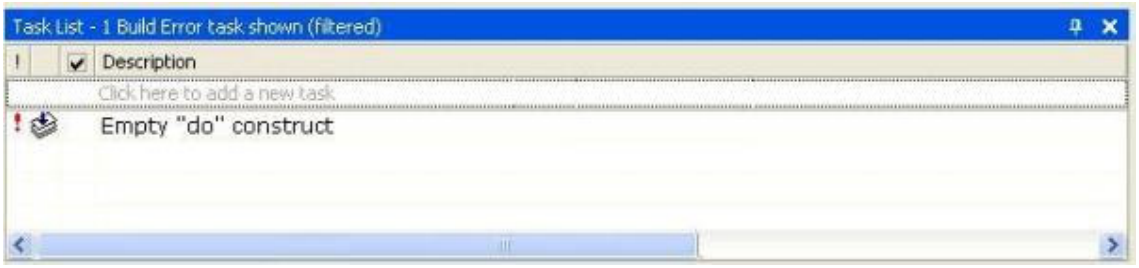

Figura 20 – Esboço de interface para o caso de uso UC08

3.3.9 [UC09] Ser direcionado para a fonte do erro a partir da *taskbar*

De modo a aumentar a produtividade do Programador Haskell, um duplo clique em um erro exibido na *taskbar* deve posicionar o cursor para o local do código-fonte que ocasionou o erro. A Figura 21 apresenta um esboço de interface para este caso de uso.

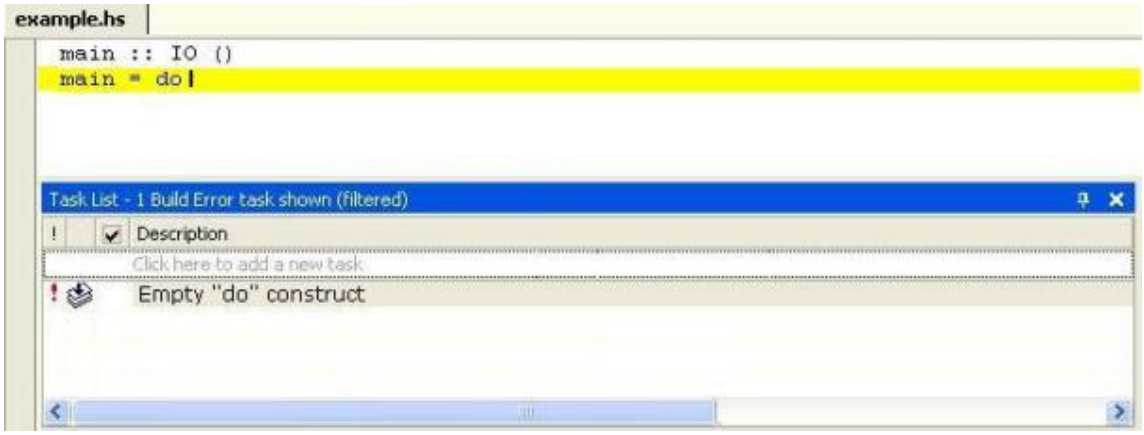

Figura 21 – Esboço de interface para o caso de uso UC09

3.3.10 [UC10] Navegar para a definição de um termo (“*go to definition*”)

O Programador Haskell deve ser capaz de, facilmente, ser redirecionado para o local em que foi declarada uma variável ou em que foi implementada uma função. Para isto, a opção “*Go to definition*” deve estar disponível no menu *popup* que é acionado quando o usuário clica com o botão direito do mouse no identificador. A Figura 22 apresenta um esboço de interface para este caso de uso.

3.3.11 [UC11] Adicionar comentários (TODO/HACK/UNDONE) à *taskbar*

O Programador Haskell deve ser capaz de visualizar, na *taskbar*, tarefas que ele mesmo (ou outro desenvolvedor) definiu como pendentes. Para adicionar tais tarefas, o programador deve fazer uso de comentários especiais (TODO/HACK/UNDONE) ao seu código-fonte. A Figura 23 apresenta um esboço de interface para este caso de uso.

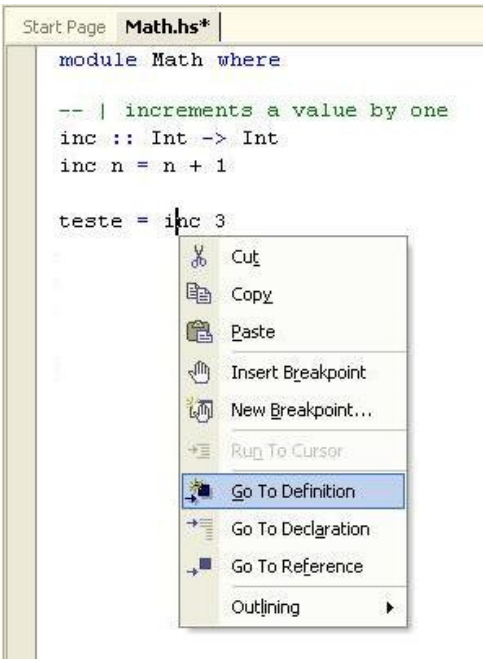

Figura 22 – Esboço de interface para o caso de uso UC10

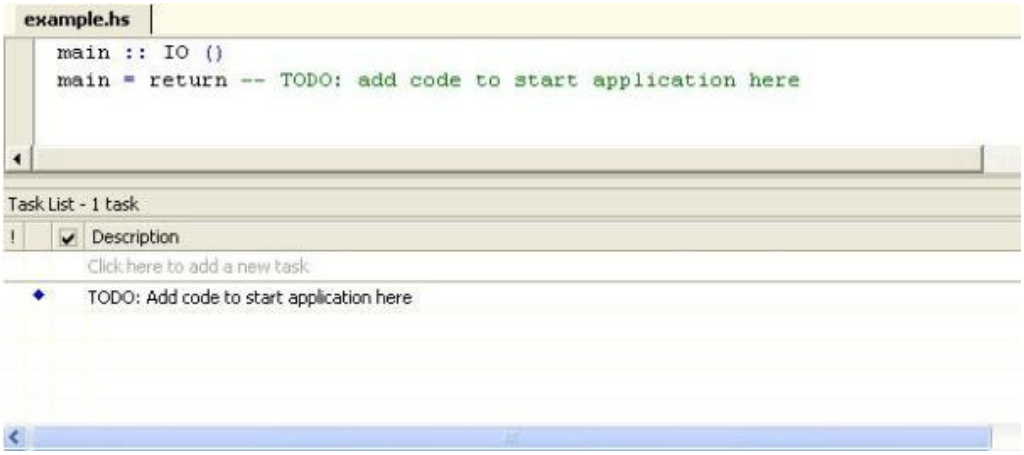

Figura 23 – Esboço de interface para o caso de uso UC11

3.3.12 [UC12] Navegar por uma *navigation bar* específica para Haskell

O Programador Haskell deve ser capaz de visualizar as funções de seu código e se mover rapidamente entre elas. A *navigation bar* é um *combo-box* que contém as funções declaradas no arquivo corrente, sendo o usuário redirecionado para o local do código que contém a declaração da função após a seleção do nome da mesma na *navigation bar*. A Figura 24 apresenta um esboço de interface para este caso de uso.

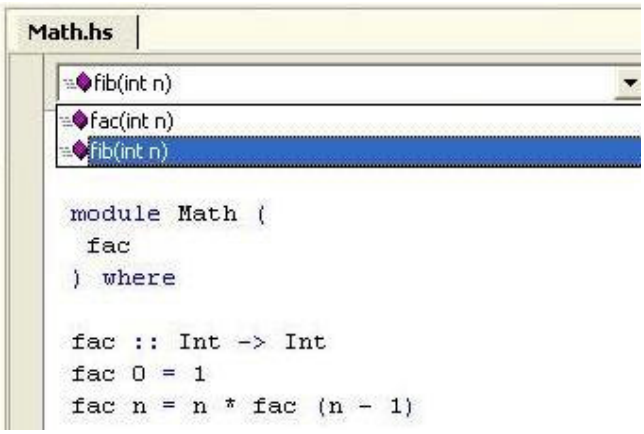

Figura 24 – Esboço de interface para o caso de uso UC12

3.3.13 [UC13] Efetuar consultas interativas ao programa via GHCi

O Programador Haskell deve ser capaz de realizar consultas à aplicação que ele está desenvolvendo chamando o interpretador GHCi, isto é, sem a necessidade de compilar a aplicação. Isto pode ser útil para a invocação de funções com o objetivo de realizar testes unitários, por exemplo. A Figura 25 apresenta um esboço de interface para este caso de uso.

Figura 25 – Esboço de interface para o caso de uso UC13

3.3.14 [UC14] Instalar o suporte a Haskell no VS.NET

O Programador Haskell deve ser capaz de instalar, no Visual Studio .NET 2003, a aplicação desenvolvida no projeto VHS-AM, fazendo com que a IDE passe a oferecer suporte à linguagem Haskell.

3.3.15 [UC15] Compilar uma aplicação Haskell

Através do comando CTRL + SHIFT + B (padrão do VS.NET), o Programador Haskell deve ser capaz de compilar a solução do VS.NET em desenvolvimento, de modo a gerar um executável e verificar se seu código-fonte está livre de erros de compilação.

3.3.16 [UC16] Executar uma aplicação Haskell

Através do comando CTRL + F5 (padrão do VS.NET), o Programador Haskell deve ser capaz de executar a aplicação em desenvolvimento, de modo a interagir com a mesma à procura de defeitos e aferir sua qualidade.

3.4 Requisitos Não-Funcionais do Projeto VHS-AM

3.4.1 [NF01] *VSIT Compliance*

O VSIT (*Visual Studio .NET Integration Test*), desenvolvido pela Microsoft, consiste em uma suíte de testes de integração. Sua função é verificar que um pacote de extensão do VS.NET não quebra nenhuma funcionalidade principal do Visual Studio .NET, garantindo que a instalação e desinstalação da aplicação desenvolvida funcionem como esperado. Portanto, a aplicação desenvolvida deve estar em concordância com o VSIT, até mesmo pelo fato de isso ser um requisito para formalizar e oficializar o trabalho diante da Microsoft.

3.4.2 [NF02] Integração com o *lexer*/*parser* do GHC

O *lexer* e o *parser* a serem utilizados pela solução devem ser os mesmos do GHC. Isto permite uma série de vantagens, como uma melhor extensibilidade da solução em relação à evolução da linguagem Haskell e uma maior facilidade na extração de informações semânticas do código elaborado pelo programador, por exemplo.

3.4.3 [NF03] Usabilidade

Uma vez que a aplicação desenvolvida esteja devidamente instalada, o usuário do VS.NET e do Assignment Manager não precisarão acessar/modificar qualquer configuração específica dessas ferramentas para poder utilizá-las com a linguagem Haskell. Em outras palavras, o usuário deve se sentir como se estivesse utilizando qualquer uma das linguagens que originalmente acompanham o VS.NET, como C# ou VB.NET, por exemplo.

3.4.4 [NF04] Estabilidade

O VS.NET e o Assignment Manager devem se comportar de maneira estável após a aplicação ser registrada. Não se deve esperar que a aplicação aborte ou trave durante a utilização de Haskell em um determinado cenário se este comportamento não é observado/esperado com outras linguagens que originalmente acompanham o VS.NET, como C# ou VB.NET, por exemplo.

3.4.5 [NF05] Performance

O VS.NET e o Assignment Manager, durante a utilização de Haskell, devem apresentar performance semelhante à utilização dessas ferramentas com outras linguagens que originalmente acompanham o VS.NET, como C# ou VB.NET, por exemplo. Entretanto, durante o carregamento da aplicação pelo VS.NET, pela primeira vez, pode-se esperar um eventual *delay*.

3.4.6 [NF06] Implantação

O processo de implantação da aplicação desenvolvida deve ser fácil e intuitivo, não exigindo experiência mais avançada do usuário. Não se deve esperar nenhum procedimento a mais do que o processo padrão normalmente utilizado para implantação/registro de pacotes de extensão no VS.NET. Um manual do usuário deve apresentar um passo-a-passo para esse processo de implantação, referenciando uma bibliografia mais completa. Adicionalmente, este manual deve especificar quais os pré-requisitos necessários para a implantação da solução, como a existência do GHC, por exemplo.

3.4.7 [NF07] Software

A aplicação deve suportar obrigatoriamente a versão 2003 do Visual Studio .NET. A princípio, a versão 2002 não faz parte do escopo do projeto.

3.4.8 [NF08] Hardware

A aplicação não deve exigir nenhuma configuração de hardware a mais do que o especificado como mínimo para o VS.NET e o Assignment Manager.

3.5 Distribuição dos Casos de Uso através dos *Releases* Internos do Projeto

Segundo o glossário da metodologia Pro.NET, um *release* interno significa “um estado conhecido do produto em desenvolvimento, obtido de maneira incremental. *Releases* internos são utilizados pela equipe para adicionar subconjuntos de funcionalidades ao produto até que ele seja completamente implementado”. Em uma analogia com a conhecida metodologia RUP (Rational Unified Process) [39], os *releases* internos da Pro.NET correspondem aos *builds* da metodologia da IBM/Rational[12].

[12] Na Pro.NET e no MSF, o termo *build* é utilizado com outro significado: uma etapa intermediária de um *release* interno, não constituindo necessariamente um estado da aplicação que possui um subconjunto de funcionalidades completo e bem-definido.

No Plano de Implementação do projeto VHS-AM (outro artefato da metodologia Pro.NET) foram definidos, inicialmente, três *releases* internos para a aplicação. A divisão dos casos de uso e dos requisitos não-funcionais entre esses três *releases* internos se deu conforme mostra a Tabela 2.

Tabela 2 – Distribuição dos casos de uso e RNFs pelos *releases* internos da aplicação

Release Interno	Objetivo	Casos de Uso	Requisitos não-funcionais
1	Validar o uso do AM com o mínimo de suporte a Haskell	UC02, UC03, UC04, UC15, UC16	NFR07, NFR08
2	Permitir um suporte maior à edição de código Haskell no VS.NET	UC01A, UC01B, UC01C, UC01D, UC05, UC06, UC07A, UC10, UC14	NFR01, NFR03, NFR04,NFR05
3	Adicionar *features* de menor prioridade e que dependem do *lexer*/*parser* do GHC	UC01E, UC01F, UC08, UC07B, UC09, UC11, UC12, UC13	NFR02, NFR06

De acordo com o cronograma inicial do projeto VHS-AM e com o prazo de entrega deste Trabalho de Graduação, era esperado que o release *interno* 2 estivesse concluído na entrega deste documento. Conforme será discutido no Capítulo 5, alguns casos de uso e requisitos não-funcionais tiveram sua priorização repensada, havendo mudanças na ordem de implementação dos mesmos e provocando atualizações no cronograma. Atualmente, o projeto VHS-AM encontra-se no processo de conclusão de seu *release* interno 2.

3.6 Demais Propostas de Casos de Uso

Algumas propostas iniciais tiveram que ficar de fora do escopo do projeto VHS-AM, devido a restrições de recursos e cronograma. Entretanto, tais propostas podem voltar a ser discutidas no projeto VHS original ou ainda no próprio projeto VHS-AM, caso o escopo inicialmente definido seja concluído antes do prazo. Uma listagem dessas propostas segue abaixo:

- Auto-identação de texto;
- Inserção de tipos da linguagem (tipos abstratos de dados, funções, etc.) em um módulo a partir de *wizards*;
- Documentação de código baseada em XML (como é o caso da linguagem C#);
- Ajuda dinâmica ligada à documentação existente na Haskell.org;
- Suporte a um modelo de código (*code model*);
- Suporte à depuração;
- Suporte a projetos Windows e Web para Haskell;
- Suporte à criação de código gerenciado pelo *Common Language Runtime* (CLR) do .NET Framework.

4. ESTENDENDO O VISUAL STUDIO .NET 2003

Uma das primeiras certezas estabelecidas para este Trabalho de Graduação e os projetos VHS/VHS-AM foi a escolha de não criar uma IDE totalmente nova para Haskell, e sim adaptar uma IDE extensível, já existente, para que a mesma oferecesse suporte à linguagem. Acredita-se que a escolha do Visual Studio .NET 2003 como IDE a ser estendida foi extremamente feliz, tendo em vista o rico conjunto de funcionalidades oferecido pelo modelo de extensão e automação desta IDE.

Apesar das versões originais do VS.NET oferecerem um amplo conjunto de ferramentas e o poder para a concretização de diferentes tarefas, como mostrou o Capítulo 2, alguns desenvolvedores ou empresas[13] podem necessitar de um nível de interação mais detalhado com a IDE [53]. Este nível pode ser proporcionado pela utilização do modelo e extensão e automação do VS.NET, que possibilita enriquecer as funcionalidades já existentes e adicionar uma infinidade de novas funcionalidades à IDE.

Este capítulo discute os diferentes mecanismos de extensão e automação do VS.NET, à luz dos casos de uso discutidos no capítulo anterior. Boa parte do conteúdo apresentado neste capítulo foi documentado e apresentado pelo aluno deste Trabalho de Graduação à equipe do projeto VHS-AM, na Fase de Planejamento do projeto, em que, segundo a metodologia Pro.NET, ocorrem discussões sobre a escolha das tecnologias a serem utilizadas para a implementação da aplicação a ser desenvolvida.

4.1 Visão Geral dos Mecanismos de Extensão do VS.NET

Antes de escrever qualquer linha de código, é necessário definir a profundidade da extensão que se deseja obter a partir do VS.NET. A Figura 26 apresenta esses diferentes níveis de extensão, comparativamente.

Na base da pirâmide, estão as possibilidades de extensão básicas, como a customização geral do ambiente e a adição de componentes na *toolbox* do VS.NET, o que será apresentado a seguir.

Um nível acima, estão as *macros* do VS.NET, conceito que tradicionalmente é aplicado na automação de tarefas rotineiras mas que, no contexto do VS.NET, aumenta o poder dos programadores através de novos recursos suportados por uma rica infra-estrutura da

[13] Á título de ilustração, as empresas IBM/Rational, CompuWare e Crystal estendem o VS.NET para o desenvolvimento de ferramentas (como o popular Crystal Reports, para geração de relatórios), enquanto empresas como a Fujitso e a ActiveState estendem o VS.NET para o suporte a novas linguagens (como o Cobol.NET [9]).

IDE, como a edição e depuração de *macros* previamente gravadas, acesso ao *object model* do VS.NET, ligação a comandos no menu e utilização do *Macro Explorer* para o gerenciamento de *macros*, por exemplo.

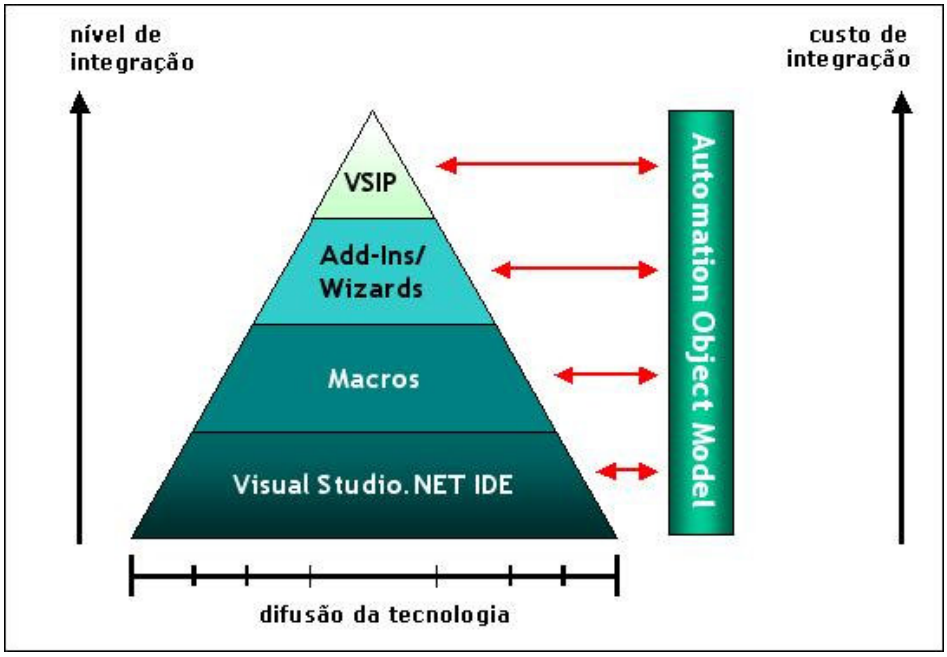

Figura 26 – Diferentes níveis de extensão do VS.NET

Os VS.NET *add-ins*, por sua vez, estão um nível acima das VS.NET *macros*. Eles permitem a integração de novos comandos a *toolbars* e menus, adição de novas janelas de ferramentas e uma automação mais poderosa de fluxos de atividades executados de maneira mais freqüente no VS.NET. No mesmo nível, os VS.NET *wizards* permitem extensão dos *templates* de projetos suportados pelo VS.NET, assim como dos "itens de projetos" que compõem esses projetos.

Por fim, no topo da pirâmide, encontra-se o VSIP (*Visual Studio Industry Partner*). Este programa, oferecido pela Microsoft a desenvolvedores, é a última palavra na área de automação/extensão do VS.NET. Através da exposição e interação com toda a arquitetura e interfaces do próprio VS.NET, o VSIP permite estender a IDE em diversos aspectos, não abordados pelos demais mecanismos de automação/extensão, como a adição de uma linguagem de programação ao ambiente, por exemplo.

Algumas informações adicionais podem ser extraídas da Figura 26. Em primeiro lugar, é intuitivo observar que, quanto maior o nível de integração oferecido por uma tecnologia, mais custosa é a implementação da integração. Além disso, quando mais especializada é a tecnologia, menor é a sua difusão, isto é, menos pessoas a utilizam. Isto é bastante negativo no sentido de que experiências e lições aprendidas por terceiros com a tecnologia

serão encontradas com maior dificuldade. Por fim, é importante observar que todas as camadas de extensão têm acesso *ao automation object model* do VS.NET. Este é um objeto exposto pelo VS.NET que permite a programadores invocarem métodos relativos ao próprio ambiente de desenvolvimento, permitindo a automação de tarefas que, normalmente, seriam feitas de maneira manual. Estes e demais conceitos serão apresentados detalhadamente nas subseções a seguir.

4.2 Automation Object Model (AOM)

4.2.1 A tecnologia

O VS.NET é composto por várias janelas de ferramentas (como a *toolbox* e a *tasklist*) e várias janelas de documento (como os editores de código-fonte), todas relacionadas entre si. O *automation object model* (AOM) do VS.NET contém objetos e interfaces da IDE que podem ser acessados programaticamente para o controle e extensão da IDE, manipulando seus comandos e janelas. O AOM é incluído no VS.NET e está publicamente documentado.

Os objetos, propriedades e interfaces do AOM provêem acesso a diversos aspectos do VS.NET. Alguns lidam com projetos e soluções, outros com a interface gráfica, outros com controle de versão, outros com comandos, outros com o depurador e outros, por fim, com configurações gerais da IDE, por exemplo. Em outras palavras, o *Automation Object Model* é composto de vários "*submodels*", cada um cobrindo uma determinada categoria de funcionalidades. Tal conceito pode ser ilustrado na Figura 27.

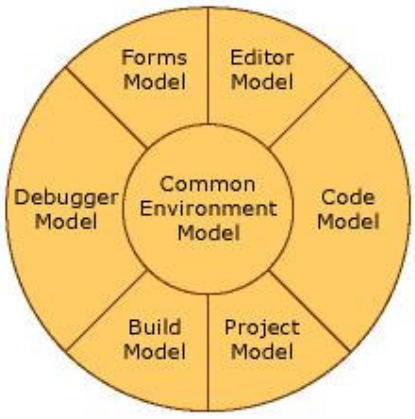

Figura 27 – *Submodels* do AOM (*fonte: VS.NET Extensibility Center* [53])

A implementação do AOM é feita na prática por um objeto conhecido como DTE (*Development Tools Extensibility*). Dado que o AOM está disponível a diferentes mecanismos de extensão do VS.NET, a maneira como é obtida a referência ao objeto DTE depende do próprio mecanismo de extensão utilizado. *Macros* utilizam uma variável global, que referencia o

objeto DTE, disponibilizada pelo Ambiente de *Macros* do VS.NET. Para um *add-in*, por sua vez, é passada, pelo VS.NET, uma referência do objeto DTE na inicialização do próprio *add-in*. Já os *scripts* utilizados em *wizards* podem referenciar o objeto DTE através de um objeto global. Aplicações implementadas a partir do VSIP, por fim, devem solicitar o objeto DTE programaticamente através de serviços e interfaces disponibilizadas pelo VS.NET [4].

4.2.2 AOM no contexto do projeto VHS-AM

A utilidade do AOM no projeto VHS-AM está diretamente relacionada aos mecanismos de extensão escolhidos. Funcionalidades implementadas com *macros*, *add-ins* e *wizards*, por exemplo, necessitam interagir consideravelmente com o objeto DTE, já que esta é a única maneira destes mecanismos de extensão acessarem programaticamente o ambiente do VS.NET. O VSIP, por outro lado, possui suas próprias interfaces de acesso ao ambiente do VS.NET, que substituem os *submodels* do AOM, sendo o mesmo utilizado em menor escala para a implementação de casos de uso nesta abordagem. Em resumo, o AOM não participou da escolha dos mecanismos de extensão a serem utilizados, sendo uma tecnologia acionada sob demanda, de acordo com as necessidades e decisões do projeto (ocasionadas por outras variáveis).

4.3 Componentes Customizados na VS.NET *Toolbox*

4.3.1 A tecnologia

A *toolbox* do Visual Studio .NET, exibida na Figura 28, permite a inserção de componentes (como botões ou caixas de texto) através de comandos simples como “arrastar e soltar”.

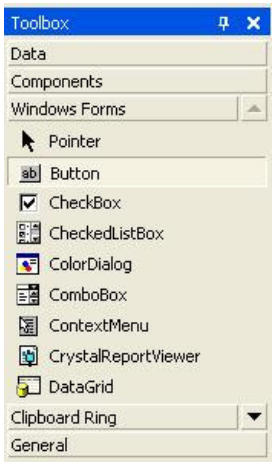

Figura 28 – *Toolbox* do VS.NET

Uma vez presente em um formulário, um componente pode ter suas propriedades e eventos acessados a partir da janela de propriedades do VS.NET, como mostra a Figura 29. É importante observar que componentes podem ter uma representação visual associada ao mesmo (como é o caso de botões e caixas de texto) ou não (como é o caso de *timers* e *datasets*).

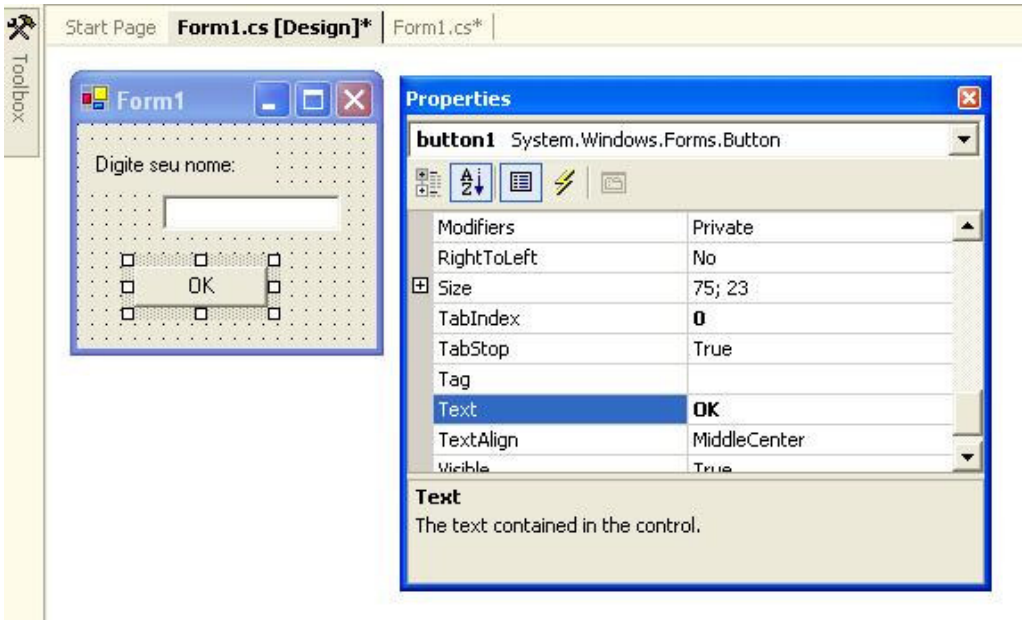

Figura 29 – Janela de propriedades do VS.NET

Através de uma linguagem gerenciada, como C# ou VB.NET, é extremamente fácil a implementação de um componente, visual ao não, a ser adicionado na *toolbox* do VS.NET: basta criar uma classe que herda da classe *Component*, definida na *class library* do .NET Framework. Uma vez compilado o projeto que contém essa classe, basta arrastar a dll resultante da compilação à própria *toolbox* para que o novo componente seja adicionado e esteja pronto para ser "arrastado e soltado" em demais projetos.

Entre outras funcionalidades, é possível especificar o ícone do componente a ser adicionado, suas propriedades, categorias para agrupar suas propriedades, descrições para cada propriedade, os eventos que o componente dispara, etc. Adicionalmente, é possível definir *designers* para os componentes visuais, que determinam como o mesmo será desenhado em um formulário Windows ou Web.

4.3.2 Componentes customizados no contexto do projeto VHS-AM

Como se esperava, componentes customizados não revelaram muita utilidade ao projeto VHS-AM. Em primeiro lugar, componentes visuais fazem sentido apenas para projetos que lidam com interfaces gráficas, o que não é o caso do tipo de projeto *Haskell Console Application*, definido no capítulo anterior. Em geral, o conceito de componentes (visuais ou não) apresentados nesta seção, além de não encontrarem uma aplicação direta nos casos de uso do projeto VHS-AM, demandariam que a equipe do projeto dominasse técnicas de implementação de componentes para linguagens não gerenciadas, o que provavelmente teria um impacto significativo no cronograma. Entretanto, não deixa de ser interessante a utilização desta tecnologia no futuro, tanto para suportar projetos Haskell que contenham interface gráfica como para a inclusão, em qualquer tipo de projeto Haskell, de recursos mais avançados como *timers*, *datasets*, componentes de impressão e de ajuda ao usuário.

4.4 *Macros*

4.4.1 A tecnologia

A utilização das VS.NET *macros* é uma das maneiras mais fáceis de estender a IDE. Apesar de pouco poderosa, esta abordagem implica em um ganho considerável de produtividade ao permitir a gravação de um conjunto de ações executadas no VS.NET e, posteriormente, a reprodução dessas ações como um único comando. É permitido também verificar e alterar o código-fonte da *macro* criada por uma gravação, sendo possível a execução de alterações pontuais nas mesmas. Devido a sua simplicidade, o conceito de *macros* pode ser utilizado como uma ferramenta para o aprendizado do próprio *automation object model*.

Macros são gerenciadas a partir de um ambiente à parte do VS.NET, chamado *VSMacros IDE*, que contém um conjunto de funcionalidades bastante rico. Este ambiente separado possui uma janela de exibição de código (*document window*) e uma variedade de janelas de ferramentas similares às do VS.NET, mas que funcionam especificamente para *macros*. A Figura 30 apresenta a *VSMacros IDE*, revelando sua semelhança com o VS.NET.

Caso o programador não sinta a necessidade de editar e depurar *macros*, abrindo mão, portanto, da *VSMacros IDE*, ainda sim ele poderá ter um controle sobre as *macros* gravadas por ele e demais *macros* disponíveis no VS.NET. Este controle é realizado pela janela de ferramentas intitulada *Macro Explorer*, que exibe todas as *macros* do sistema, permitindo sua deletação, renomeação e execução, além de disparar a *VSMacros IDE* caso esta seja a vontade do usuário. O *Macro Explorer* é apresentado na Figura 31.

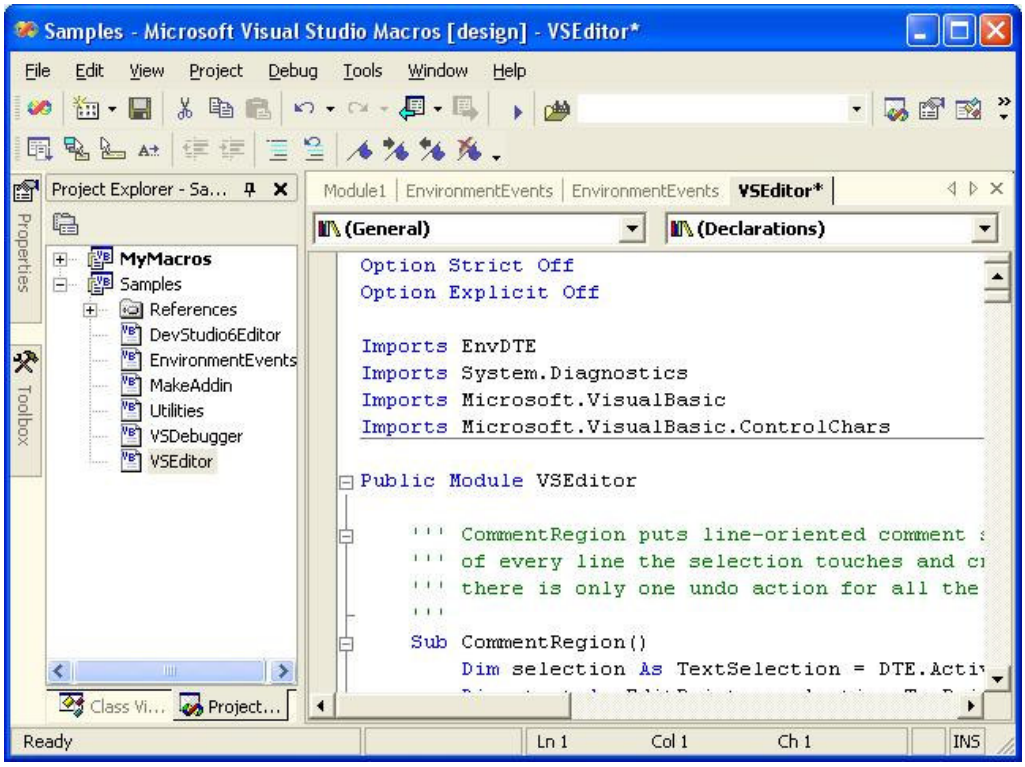

Figura 30 – VSMacros IDE

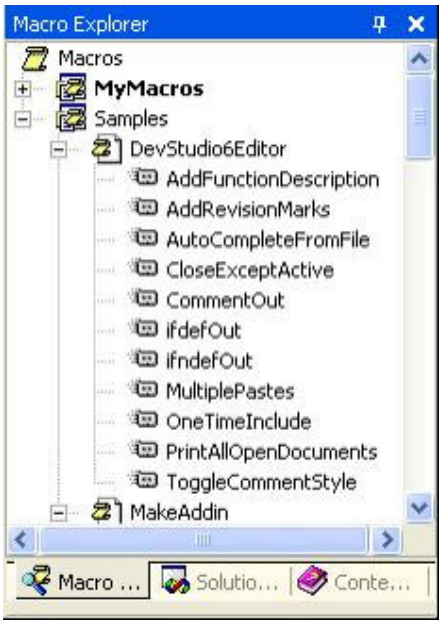

Figura 31 – Macro Explorer

4.4.2 *Macros* no contexto do projeto VHS-AM

Sem dúvida, *macros* são úteis, ágeis e produtivas. Como elas não estão associadas a uma linguagem específica, nada impede o seu uso pelo programador Haskell (isto é, esta funcionalidade é dada gratuitamente pelo VS.NET). Entretanto, *macros* possuem muitas limitações que as impedem de ser consideradas na implementação das funcionalidades expostas no capítulo anterior. As principais limitações que conduziram a tal conclusão são:

- Não é possível construir interfaces gráficas com *macros*;
- Não é possível criar janelas de ferramentas customizadas com *macros*;
- Não é possível habilitar e desabilitar dinamicamente itens em menus e barras de ferramentas com *macros*;
- Não é possível criar um instalador para funcionalidades implementadas com *macros*;
- *Macros* no VS.NET são obrigatoriamente escritas em VB/VB.NET, linguagens com as quais os integrantes do projeto VHS-AM não possuem muita intimidade.

4.5 *Add-Ins*

4.5.1 A tecnologia

De modo a superar as limitações das *macros*, o VS.NET oferece uma tecnologia mais poderosa (e, conseqüentemente, um pouco mais complexa) para a extensão e integração do ambiente de desenvolvimento: os VS.NET *add-ins*. Eles são aplicações que se acoplam à IDE e são utilizadas dentro da mesma, comunicando-se com o ambiente através do *Automation Object Model*.

Add-ins são componentes COM[14], o que implica em um esforço adicional para sua implementação, pois é necessário compilar e instalar *add-ins* antes que os mesmos possam ser utilizados (ao contrário das *macros*, que são escritas e executadas mais diretamente). As vantagens da utilização de *add-ins*, entretanto, são evidentes. Além de ter acesso a uma API mais poderosa do que as *macros*, fazendo um melhor uso do AOM do VS.NET, *add-ins* são mais fáceis de serem distribuídos e não possuem seu código exposto (como é o caso das *macros*), pois eles são compilados em código binário. Isto evita alterações inadvertidas ou maliciosas por parte de terceiros, protegendo a propriedade intelectual do desenvolvedor.

Add-ins são ideais para desenvolvedores e empresas que desejem integrar aplicações de média complexidade ao VS.NET, dando a impressão, ao usuário final, de que essas aplicações são nativas da IDE. Alguns exemplos das possibilidades proporcionadas pelos *add-ins*, ausentes nas VS.NET *macros*, são:

---

[14] A tecnologia COM será abordada em mais detalhes no Capítulo 5.

- Criação de janelas de ferramentas (*tool windows*) que se comportam como qualquer outra janela de ferramentas do VS.NET;
- Habilitação e desabilitação dinâmica de comandos em menus e barras de comandos do VS.NET;
- Criação de páginas de propriedades customizadas para a janela *Options*, acessível a partir do menu *Tools* da IDE (um exemplo é apresentado na Figura 32);
- Adição de informações de contato e descritivas na janela *About*, presente no *Help* do VS.NET.

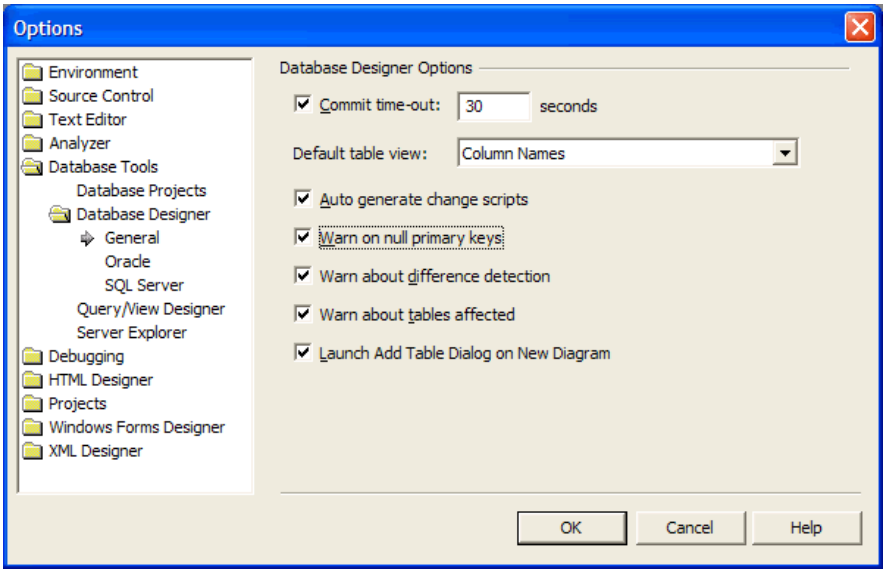

Figura 32 – Exemplo de página de propriedades da janela *Options*

O VS.NET oferece um rico suporte à criação de *add-ins*. Ao solicitar a criação de um novo projeto ao VS.NET, o desenvolvedor pode escolher, além dos tradicionais tipos *Console Application*, *Windows Application* e *Web Service*, por exemplo, um tipo de projeto específico para a criação de *add-ins*. Esta opção, quando selecionada pelo desenvolvedor, cria um *framework* contendo os projetos e as classes necessárias para iniciar o desenvolvimento do *add-in*, incluindo, inclusive, um projeto de instalação para o mesmo. Resta ao desenvolvedor, portanto, apenas preencher os "ganchos" deixados propositalmente por esse *framework*, de modo a adicionar funcionalidades a seu *add-in*.

4.5.2 Add-ins no contexto do projeto VHS-AM

A utilização de *add-ins* para a implementação de casos de uso menos complexos do VS.NET, a princípio, pareceu um caminho a ser seguido, principalmente pela presença do *framework* para a criação de *add-ins* no VS.NET, que não encontra similar na versão original do VSIP.

Entretanto, com o surgimento da recente versão “Extras” do VSIP, publicada oficialmente em março de 2004, o VS.NET passou a oferecer também um *framework* para abordagens de implementação baseadas no VSIP. Este novo *framework* parece ter se adequado perfeitamente aos casos de uso do projeto VHS-AM que seriam anteriormente implementados por *add-ins*. Por exemplo, percebeu-se que a janela de ferramentas a ser criada para implementar a integração do VS.NET com o interpretador GHCi (UC13), permitindo ao usuário a realização de consultas ao código em desenvolvimento, teria um custo de implementação praticamente semelhante, independentemente da escolha entre *add-ins* e o VSIP Extras. Adicionalmente, uma vez que a infra-estrutura de desenvolvimento estivesse consolidada e a tecnologia dominada, a utilização do VSIP traria vantagens adicionais devido ao seu maior poder de extensão e integração.

Dois outros itens diminuíram a necessidade pela utilização de *add-ins* no projeto VHS-AM. O primeiro diz respeito à incapacidade desta tecnologia em criar novos tipos de projeto e estender linguagens, ao contrário do VSIP. Isto descarta a utilização de *add-ins* para a implementação de grande parte dos casos de uso. O segundo item se refere à implementação dos demais casos de uso, como é o caso da invocação do GHCi para consultas (UC13) e a utilização de um *module viewer* para Haskell (UC06). Nestes casos, temeu-se que a diferença das estratégias de implementação desses casos de uso com os demais provocasse incompatibilidade ou um *overhead* de programação, principalmente pelo alto grau de relacionamento entre os casos de uso em geral.

Entretanto, como as necessidades da estratégia de implementação de um caso de uso tornam-se mais claras apenas quando o mesmo é abordado mais profundamente, ainda há a possibilidade de que *add-ins* venham a ser úteis na implementação de alguns casos de uso restantes do projeto, embora isto não seja esperado.

4.6 *Wizards*

4.6.1 A tecnologia

VS.NET *wizards* constituem uma tecnologia cujo principal objetivo é conduzir o desenvolvedor através de passos automatizados para a conclusão de tarefas complexas de serem executadas manualmente. Exemplos de tais tarefas são a adição de um projeto a uma solução e a adição de um Web Service a um projeto. Uma vez concluída a tarefa, a missão do *wizard* está cumprida e ele não mais é utilizado até que a tarefa seja solicitada novamente.

Wizards são tradicionalmente conhecidos como janelas de diálogo do tipo “*Next, Next, Finish*”, através das quais o usuário insere um conjunto de entradas e, ao fim do *wizard*, obtém uma saída específica. Este conceito é um pouco mais elaborado no VS.NET, que divide *wizards* em três categorias.

Wizards de projeto são executados quando o usuário seleciona um *template* de projeto a partir da janela "*New Project*" ou "*Add Project*". O *wizard* pode, então, apresentar uma interface gráfica na qual o desenvolvedor especifica as opções para o novo projeto a ser criado. Em seguida, o *wizard* copia alguns arquivos de *template* e os adiciona ao projeto. O *framework* para *add-ins* do VS.NET, apresentado na seção anterior, utiliza um *wizard* de projeto. Uma página da interface gráfica deste *wizard* é exibida na Figura 33.

Figura 33 – *Wizard* de projeto utilizado pelo *framework* de add-ins do VS.NET

Wizards de item, por sua vez, adicionam um novo item ao projeto corrente. Também é possível a exibição de uma interface gráfica neste tipo de *wizard*, embora isto seja menos comum. Por fim, *wizards* de contexto inserem texto um determinado arquivo, como a adição de um novo método a uma classe. A presença de uma interface gráfica neste tipo de *wizard*, também conhecida como *sekeleton insertion*, pode ser útil para orientar programadores que ainda não estejam familiarizados com a sintaxe da linguagem. A Figura 34 ilustra alguns tipos de *skeleton insertions* que podem ser utilizados por classes C#.

Para implantar um *wizard*, é necessário realizar as manipulações adequadas no registro do Windows e, adicionalmente, saber lidar com arquivos de configuração específicos do VS.NET para *wizards*, que se dividem em dois tipos. Os arquivos **.vsdir** definem uma lista de *wizards* (de projeto, item ou contexto) suportados por uma determinada linguagem, especificando informações como o ícone do *wizard*, sua ordenação em relação a demais *wizards*, seu nome e sua descrição, por exemplo. Os arquivos de configuração **.vsz**, por sua vez, são referenciados pelos .vsidr e indicam qual o *wizard engine*[15] utilizado (o padrão do VS.NET ou um implementado pelo desenvolvedor), passando parâmetros ao mesmo.

[15] Um *wizard engine* possui três tarefas: (1) montar a GUI do *wizard* para o usuário, (2) coletar as entradas do usuário durante a execução do *wizard* e (3) disparar um script para executar tarefas específicas ao fim do *wizard*, como a cópia de *templates* de arquivos, por exemplo.

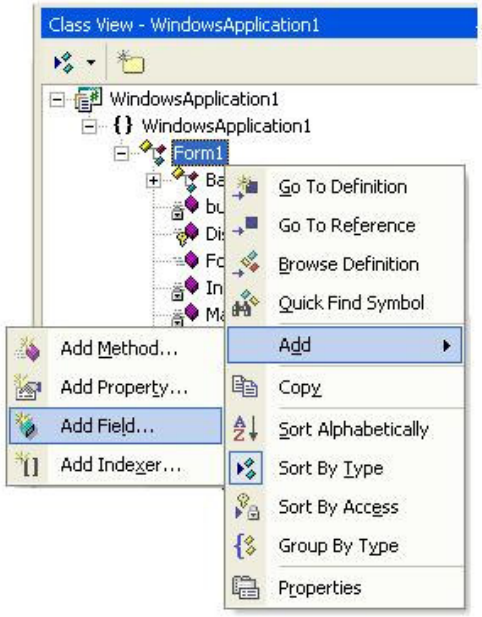

Figura 34 – *Wizards* de contexo (*skeleton insertions*) para uma classe C#

4.6.2 *Wizards* no contexto do projeto VHS-AM

Wizards de projeto são uma excelente maneira para adicionar novos *templates* de projeto às linguagens originais do VS.NET. Entretanto, o projeto VHS-AM demanda novos tipos de projeto, para novas linguagens, e não apenas novos *templates* de projeto. Dessa forma, o suporte a projetos oferecido pelo VSIP se apresentou como uma melhor opção.

Wizards de item, entretanto, revelaram-se como a maneira mais simples de implementar o caso de uso referente à adição de um novo módulo Haskell a um projeto do tipo *Haskell Console Application*. Isto acontece porque o *wizard* necessário neste caso de uso não necessita de um maior poder integração, não havendo nem a necessidade de interface gráfica para a execução desse *wizard*. Seria necessário, apenas, manipular corretamente as chaves necessárias no registro do Windows e especificar a localização do arquivo de *template* do módulo Haskell.

Wizards de contexto, por sua vez, não mostraram nenhuma aplicação para o projeto VHS-AM, pois nenhum caso de uso especificado se refere a *skeleton insertion*, isto é, à adição de tipos da linguagem (como funções, tipos abstratos de dados, etc.) a um módulo Haskell. Entretanto, este tipo de *wizard* será certamente utilizado no futuro, quando as propostas levantadas para o projeto VHS, referentes a *skeleton insertion*, forem implementadas.

4.7 Visual Studio Industry Partner (VSIP) Program

4.7.1 A tecnologia

As seções anteriores mostraram diversos mecanismos de automação, extensão e integração do VS.NET. Apesar de suas vantagens, esses mecanismos, de uma maneira geral, não atendem às necessidades do projeto VHS-AM, sendo necessário um nível mais profundo de integração.

O VSIP é um programa lançado pela Microsoft, significando *Visual Studio Industry Partner*. Desse modo, dando uma maior atenção à nomenclatura utilizada pela Microsoft, o termo VSIP se refere à pessoa ou empresa que está inscrita no programa, e não ao programa em si, sendo este último corretamente referenciado como VSIP Program. Entretanto, esta nomenclatura é recente, pois VSIP significava *Visual Studio Integration Program* ainda no segundo semestre de 2003. Este documento se referirá ao programa como simplesmente VSIP, termo ainda comumente usado para designar o mesmo e que contribuirá para não sobrecarregar o texto aqui apresentado.

Apesar de gratuito, é necessário concordar com uma licença para a inscrição no VSIP [49]. O aluno deste Trabalho de Graduação foi o responsável por ler integralmente esta licença para levantar, eventualmente, discussões sobre a não-conformidade dos projetos VHS/VHS-AM com algum item da licença.

Fazer parte do VSIP significa ter acesso às interfaces utilizadas no próprio desenvolvimento do VS.NET, usando o *VS.NET Integration SDK*. Através da implementação de componentes COM a partir destas interfaces, é possível criar *Visual Studio Packages* (ou *VSPackages*), que trocam serviços e dados com a IDE e seus "*built-in VSPackages*" (como o *CSharpProjectPackage*, *HtmlEditorPackage*, etc.). Um *VSPackage* é um componente COM, registrado de uma maneira especial, que publica serviços através de entradas no registro do Windows. O VS.NET carrega *VSPackages* automaticamente quando seus serviços são necessários. Apesar de todos os *VSPackages* implementarem a mesma interface *IVsPackage*, o conjunto de funcionalidades oferecido por cada um pode variar bastante.

VSPackages permitem a implementação de funcionalidades antes impossíveis com *macros* e *add-ins*. Obviamente, do mesmo modo que *add-ins* são mais custosos de se implementar do que *macros*, a construção de um *VSPackage* é muito mais complexa do que uma *macro* ou *add-in*. Adicionalmente, enquanto o usuário pode definir quais *add-ins* ele deseja habilitar e desabilitar no VS.NET, a única maneira de desabilitar um *VSPackage* é desinstalá-lo, removendo-o do registro do Windows, o que pode tornar o processo de desenvolvimento ainda mais complicado. A Tabela 3 exibe uma comparação das funcionalidades suportadas por *add-ins*, *macros* e *VSPackages*.

Tabela 3 – Funcionalidades suportadas por *macros*, *add-ins* e *VSPackages* [26]

Funcionalidade	*Macros*	*Add-ins*	*VSPackages*
Manipulação do objeto DTE	Sim	Sim	Sim
Criação de janelas de ferramentas	Não	Sim	Sim
Inserção de um comando de menu	Não	Sim	Sim
Criação de página de propriedades customizada na janela *Options*	Não	Sim	Sim
Informações na janela *About*	Não	Sim	Sim
Informações na tela inicial do VS.NET (*splash screen*)	Não	Não	Sim
Criar um novo tipo de projeto	Não	Não	Sim
Fazer parte de um *build*	Não	Não	Sim
Criar um depurador	Não	Não	Sim
Criar um editor de texto	Não	Não	Sim
Criar um editor gráfico	Não	Não	Sim
Adicionar dados no *Server Explorer*	Não	Não	Sim
Adicionar parâmetros ao comando devenv.exe	Não	Não	Sim
Adicionar suporte à edição (como *IntelliSense*) ao editor	Não	Não	Sim
Desenvolver utilizando uma linguagem gerenciada	Sim (só VB.NET)	Sim	Sim (VSIP Extras)

Atualmente, o VSIP encontra-se em sua versão “Extras”, que permite a criação de *VSPackages* a partir de uma linguagem gerenciada e oferece *wizards* para a criação de um *framework* contendo uma implementação básica de um *VSPackage*. Toda a equipe do projeto VHS-AM participou como testadora das versões “*VSIP Extras Beta*” e “*VSIP Extras Refresh*“, liberadas entre a versão original no VSIP e a versão atual. Alguns dos *bugs* identificados consistem na inconsistência da documentação do VSIP a partir da versão *Refresh*. Esta é uma das dificuldades encontradas a serem discutidas no Capítulo 5.

4.7.2 O VSIP no contexto do projeto VHS-AM

Cruzando as informações da Tabela 3 com o questionário *Choosing the Appropriate Automation Approach* [53], do *Microsoft Visual Studio Develper Center*, chega-se à conclusão de que o VSIP é realmente a abordagem mais promissora ao projeto VHS-AM, satisfazendo a implementação de vários de seus casos de uso.

Entretanto, como mostra o capítulo a seguir, diversas questões mais complexas relativas à estratégia de implementação, principalmente em relação ao VSIP, precisaram ser abordadas. O surgimento dessas questões se revelou um processo natural no projeto, à medida que a equipe adquiria mais intimidade com os conceitos, tecnologias e ferramentas necessárias à sua realização.

5. ABORDAGEM DE DESENVOLVIMENTO E RESULTADOS OBTIDOS

Dizer apenas que o VSIP foi a estratégia utilizada para a extensão e integração do VS.NET com Haskell e o Assignment Manager é insuficiente, não apenas para o leitor deste documento como também para a própria equipe do projeto. Isto se deve ao fato de que o desenvolvimento do projeto e a implementação de seus casos de uso e requisitos não-funcionais, apresentados no Capítulo 3, envolvem um conjunto diversificado de conceitos, tecnologias e ferramentas, algumas delas não discutidas nos capítulos anteriores. Este capítulo apresenta a evolução da abordagem de implementação utilizada pela equipe do projeto VHS-AM, assim como as tecnologias por trás dessa evolução e a infra-estrutura necessária para suportar a Fase de Desenvolvimento (na qual, segundo a metodologia Pro.NET, o escopo da aplicação a ser desenvolvida é implementado). Os resultados obtidos nesta fase, até então, também serão apresentados, contrapondo as conquistas com as dificuldades encontradas.

5.1 Uma Escolha Dentro da Escolha

Ao se aprofundar em estudos mais técnicos sobre o VSIP, muitos dos quais foram documentados pelo aluno deste Trabalho de Graduação e apresentados aos demais integrantes do projeto VHS-AM, encontrou-se um ponto de decisão crítico para a abordagem de desenvolvimento a ser seguida: o próprio VSIP, escolhido como estratégia de extensão e integração do VS.NET, oferece três diferentes possibilidades para a criação de suporte à edição de uma nova linguagem. A Figura 35 ilustra essas três diferentes abordagens, representadas pelos pequenos círculos amarelos.

Para o entendimento das abordagens, é necessário compreender, inicialmente, que o VSIP oferece vários conjuntos de interfaces, cada um relacionado com uma funcionalidade a ser implementada. Para implementar o suporte a um novo tipo de projeto, por exemplo, é necessária a implementação das interfaces *IVsProject*, *IVsProjectFactory* e *IVsHierarchy*, entre muitas outras. O conjunto de interfaces que é necessário implementar para se obter o suporte à edição de uma nova linguagem, por sua vez, chama-se *language service interfaces*.

A primeira abordagem para implementar o suporte à edição a partir do VSIP consiste em implementar diretamente as *language service interfaces*. Esta é a abordagem mais poderosa de implementação, porém ela exige um esforço considerável do programador.

Uma segunda alternativa consiste em implementar a interface *IBabelService*, exposta por um *VSPackage* intitulado **Babel**, que acompanha o VSIP e implementa as *language service interfaces*, expondo-as para o desenvolvedor em um nível maior de abstração. O Babel é considerado um *helper framework*, que permite a implementação das funcionalidades bá-

sicas de suporte à edição (como o exigido pelos casos de uso do projeto VHS-AM) sem a necessidade do entendimento das complexas *language service interfaces*. De modo a poder utilizar esta abordagem, entretanto, é necessária a presença de um compilador (com um analisador léxico e um *parser*) que possa prover informações necessárias ao Babel e ao VS.NET, como a identificação de espaços em branco, parênteses e comentários, além de ser capaz de se recuperar de erros encontrados e prover uma árvore sintática completa.

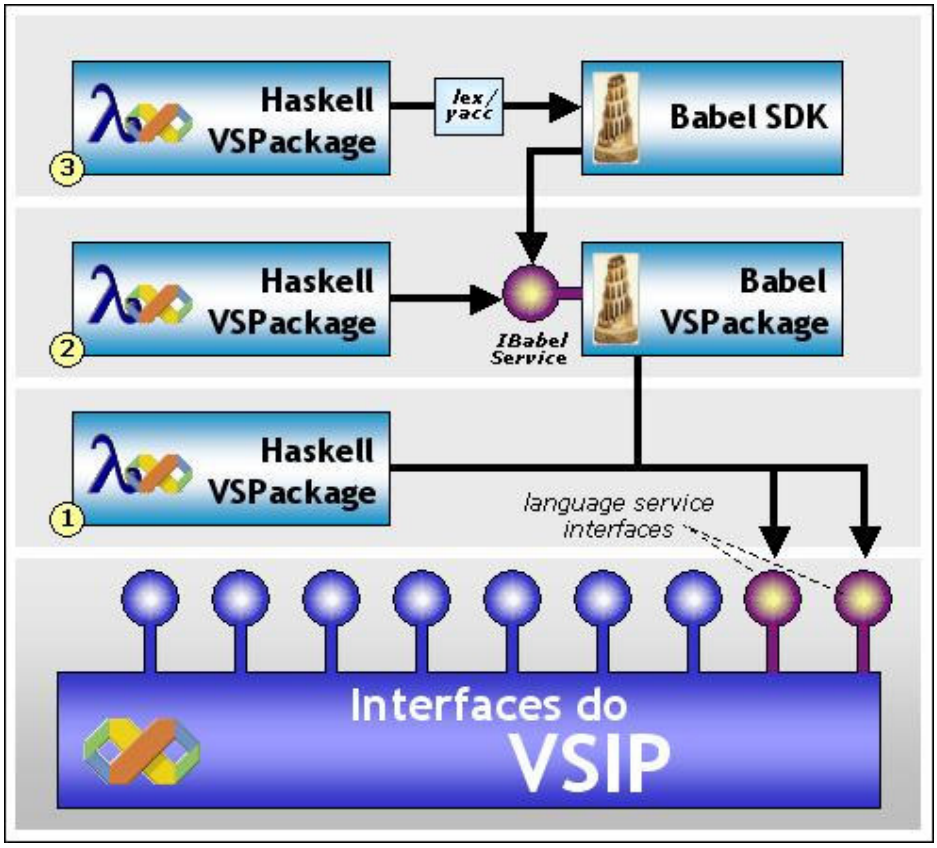

Figura 35 – Alternativas de implementação de suporte à linguagem com o VSIP

A terceira abordagem consiste em utilizar uma implementação *default* da interface *IBabelService*, disponibilizada pelo Babel SDK. Esta abordagem requer apenas que sejam especificados um analisador léxico e um *parser*, o que geralmente é feito através das ferramentas *lex* e *yacc* [83] caso o compilador da linguagem seja incapaz de interagir com o VS.NET, como é o caso do GHC.

A primeira abordagem foi prontamente descartada pela equipe do projeto, pois ela demandaria um excesso de trabalho por sua dificuldade de compreensão e implementação. Além disso, seria necessária a implementação de um *parser* para Haskell em C# (linguagem mais viável para a criação desse *VSPackage*), tarefa nunca antes realizada e que poderia comprometer consideravelmente o cronograma do projeto.

A equipe do projeto ficou tentada pela utilização da terceira abordagem, por ser a mais simples de todas e constituir uma maior chance de sucesso. A criação de um *parser* para Haskell em *lex*/*yacc* não seria necessária, pois já existe um implementado (e disponível) para o interpretador Hugs (ver Capítulo 2). Entretanto, esta abordagem revelou uma série de desvantagens após uma análise mais profunda:

- O *parser* utilizado pelo compilador GHC, o Happy [33], seria diferente do *parser* utilizado para o suporte a Haskell no VS.NET. Desse modo, haveria um trabalho adicional em manter o sincronismo entre ambos;
- Não seria possível tirar proveito do analisador de tipos do GHC (disponível através de seu *parser*), contribuindo para que a extração de mensagens de erro de tipo fosse uma tarefa mais complexa;
- A implementação dos casos de uso do projeto se daria em C (por questões de integração com o *lex*/*yacc*), e não em Haskell. Isto é uma contradição filosófica para o projeto: se ele acredita em Haskell como linguagem de programação e deseja ampliar seu campo de aplicação, então ele deveria ser o primeiro a dar o exemplo e utilizar Haskell em sua implementação.

Além das desvantagens apresentadas acima, a escolha dessa terceira abordagem inutilizaria o trabalho até então desenvolvido no projeto VHS original, no qual o pesquisador Simon Marlow defendeu e utilizou a segunda abordagem. Este trabalho é apresentado na seção a seguir.

5.2 O "*tarball* de Simon Marlow"

Como informado no Capítulo 1, o projeto VHS já vinha realizando alguns avanços em relação à integração do Visual Studio .NET 2002 com Haskell. Em outubro de 2003, quando o projeto VHS-AM ainda não existia, o pesquisador Simon Marlow publicou, para os membros do projeto VHS, um *tarball*[16] contendo uma implementação básica dessa integração.

Inicialmente, o aluno deste Trabalho de Graduação se encarregou da realização de testes de caixa-preta[17] no código do *tarball*, com o objetivo de verificar a corretude de suas funcionalidades. Antes mesmo da execução desses testes, entretanto, foi possível verificar que a documentação relativa à instalação do *tarball* estava, em alguns pontos, ambígua ou incorreta. Desse modo, foi documentado no *site* do projeto VHS o procedimento correto que levaria à instalação bem sucedida do *tarball*. Tal procedimento foi incorporado posteriormen-

[16] Conjunto de arquivos compactados através do programa *tar* do Unix/GNU [28].

[17] Ao contrário de testes de caixa-branca, um teste de caixa-preta não se preocupa com a estrutura interna do item a ser testado, sendo focado no resultado obtido a partir de uma determinada entrada. [19]

te pelo próprio Simon Marlow e até hoje faz parte do arquivo de instruções para a instalação da aplicação.

A compilação do código do *tarball* exigia a compilação de algumas ferramentas disponíveis no repositório oficial da Haskell.org, acessível através do utilitário CVS [84]. Alguns exemplos dessas ferramentas são o analisador léxico Alex [30], o *parser* Happy [33] e a ferramenta HDirect, apresentada a seguir. Infelizmente, contrariando as boas práticas da Engenharia de Software, era comum que a versão mais recente do código-fonte de algumas dessas ferramentas simplesmente não compilasse (fato popularmente conhecido como "quebra do build"). O próprio aluno deste Trabalho de Graduação, por algumas vezes, deparou-se com problemas desse tipo em relação ao HDirect, sendo necessária uma notificação a seus mantenedores para que as devidas correções fossem efetuadas. Adicionalmente, o próprio processo de compilação e configuração das ferramentas, mesmo quando elas não apresentavam uma "quebra de *build*", consistiu em um desafio, devido à grande quantidade de passos e configurações necessárias a serem executadas pelo desenvolvedor. Algumas citações abaixo, extraídas da documentação de ferramentas do repositório da Haskell.org, retratam tal dificuldade:

- "*make is great if everything works [...] Our goal is to make this happen often, but somehow it often doesn't; instead some weird error message eventually emerges from the bowels of a directory you didn't know existed.*" [32]
- "*Debugging Makefiles is something of a black art […]*" [32]
- "*The Windows situation for building GHC is rather confusing.*" [31]
- "*There are several strange things about ssh on Windows that you need to know.*" [31]
- "*If you are paranoid, delete config.cache if it exists.*" [31]
- "*The latest copy of the building guide is in fptools/docs/building, but it needs to be built from source.*" *[Simon Marlow, em resposta a uma dúvida sobre o processo de build da Haskell.org]*

Complicando ainda mais a situação, versões recentes algumas ferramentas, como o MinGW [66], o gcc [25] e o próprio GHC quebraram a compatibilidade com o código-fonte do *tarball*. Após um longo processo de tentativa e erro, finalmente foi dominada com segurança a técnica de compilação das ferramentas do repositório da Haskell.org. Analogamente, a resolução de eventuais dificuldades se tornou menos problemática, devido à experiência com problemas similares anteriores. Essas dificuldades encontradas foram registradas no documento de Postmortem do projeto, que, de acordo com a metodologia Pro.NET, deve ser utilizado para armazenar as principais conquistas, desafios e lições aprendidas vivenciadas pela equipe.

A superação das dificuldades acima permitiu que o *tarball* disponibilizado por Marlow fosse finalmente compilado, com o objetivo da realização de testes de caixa-preta. Entretanto, confirmando um problema anteriormente ocorrido com o professor André Santos, mas

não com o próprio Simon Marlow, o *tarball* compilado pelo aluno deste Trabalho de Graduação se comportou de maneira extremamente instável, abortando logo após ser carregado pelo VS.NET. Um *workaround* para o problema foi identificado pelo próprio professor André Santos. A invocação do VS.NET deveria acontecer acompanhada de uma solicitação ao *prompt* de comando: o redirecionamento do *output* de *standart error* (stderr) para um arquivo. Isto evitou que o VS.NET abortasse e o *tarball* pôde, finalmente, ser testado.

Os testes realizados revelaram que o *tarball*, apesar de ainda mostrar algumas deficiências de performance e instabilidade (superadas atualmente), já apresentava com sucesso a funcionalidade de *syntax coloring*, implementada através da segunda das abordagens disponibilizadas pelo Babel (Figura 35). Uma vez completos os testes de caixa-preta, não apenas o código do *tarball* como também o seu *build system*[18] passaram ser estudados em maior profundidade. Essa etapa seguinte coincidiu com o surgimento oficial do projeto VHS-AM, precisando ser definido, portanto, um processo sistemático para a adição de novas funcionalidades à aplicação e uma infra-estrutura para o ambiente de desenvolvimento do projeto.

5.3 Novas Tecnologias e Ferramentas a Serem Dominadas

A primeira dificuldade encontrada na identificação do que era necessário para a adição de novas funcionalidades à aplicação consistiu na não-familiarização do aluno deste Trabalho de Graduação com a tecnologia COM, indispensável para o entendimento da arquitetura de implementação presente no *tarball* e em seu *build system*. Desse modo, um estudo sobre a tecnologia foi realizado, sendo o resultado do mesmo documentado na Base de Conhecimento do projeto e apresentado à equipe de desenvolvimento.

5.3.1 A Tecnologia COM

No início, aplicações eram formadas por apenas uma única peça, geralmente um arquivo binário, sendo, portanto, monolíticas. Com o tempo, percebeu-se que esta abordagem possuía uma séria desvantagem: novas versões da aplicação necessitavam de uma re-compilação e re-implantação, revelando seu caráter estático e inflexível uma vez implantada no ambiente de produção.

De modo a resolver tal problema, foi criado o conceito de “programação em componentes”. Tal inovação permitiu que a evolução das aplicações se desse de maneira mais produtiva e eficiente, além de viabilizar uma maior customização de aplicações, o surgimen-

[18] Descreve como o código-fonte de uma aplicação é compilado, dependência por dependência, até que seja atingido o executável final.

to de bibliotecas de componentes para agilizar o desenvolvimento e a distribuição em rede de várias partes de uma mesma aplicação.

Neste contexto, a tecnologia COM surgiu como uma alternativa para padronizar a programação em componentes. COM é uma especificação, que define como construir componentes que podem ser substituídos dinamicamente. COM, portanto, especifica regras a serem seguidas por componentes e clientes desses componentes, permitindo que eles operem de forma conjunta. Neste trabalho, um componente criado de acordo com a especificação COM é chamado de "componente COM".

Os componentes COM são constituídos de código executável, distribuído na forma de bibliotecas de ligação dinâmica (DLLs ou *dynamic link libraries*) ou arquivos executáveis (EXEs). Estes componentes podem ser acoplados dinamicamente a uma aplicação e encapsulam seus detalhes de implementação, publicando apenas quais são os serviços que implementam. Eles podem ser implementados independente de linguagem, atualizados sem "quebrar" seus clientes e realocados transparentemente em rede (isto é, o componente em um sistema remoto é tratado da mesma maneira que um componente no sistema local). Por fim, componentes COM anunciam sua existência também de uma maneira padrão, permitindo que clientes encontrem dinamicamente os componentes que precisam utilizar.

O principal conceito por trás da utilização de componentes COM são as interfaces COM. Grosso modo, uma interface COM consiste no conjunto de serviços (métodos ou funções) exportadas por um componente COM, mas que não entra em detalhes de implementação. Dessa forma, o desenvolvimento de um componente COM consiste prioritariamente em definir os serviços oferecidos e especificá-los como interfaces COM para, em seguida, implementá-los. É interessante observar, portanto, que várias interfaces podem ser implementadas por um mesmo componente COM.

A diferença de interfaces COM para o conceito tradicional de interfaces é que interfaces COM possuem uma estrutura de memória bem-definida, contendo um array de ponteiros de função. Cada elemento deste array, portanto, contém o endereço para uma função implementada pelo componente. Entretanto, os benefícios providos pelo conceito de interfaces tradicionais estão da mesma forma presentes em interfaces COM, como o reuso de aplicações e polimorfismo.

Dois outros conceitos são essenciais para o entendimento da tecnologia COM: *stubs e proxies*. Um *proxy* é um componente COM que atua no mesmo processo de seu cliente (estando portanto em uma DLL) simulando e se comunicando com outros componentes COM em processos diferentes. Um *stub*, por sua vez, é um componente COM (DLL) que atua no mesmo processo de um outro componente COM (EXE) com o objetivo de empacotar e desempacotar dados de/para o cliente. A Figura 36 esclarece melhor os conceitos.

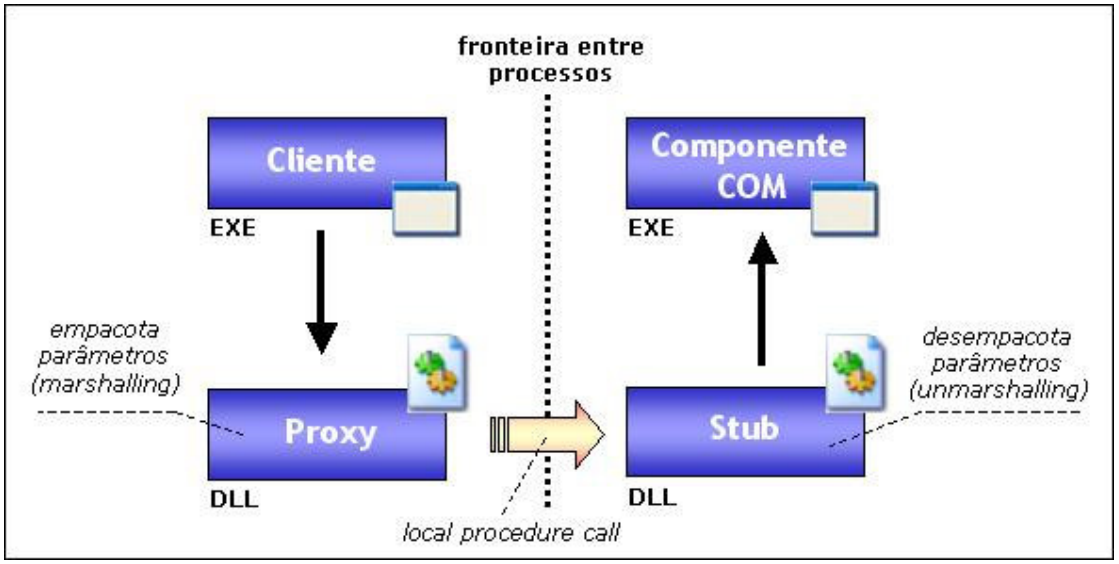

Figura 36 – Fluxo de eventos em uma chamada de um cliente a um componente COM

A implementação de componentes COM é suportada pelo conceito de IDL [85], ou *Interface Definition Language*. A IDL é uma especificação que permite descrever interfaces de maneira independente de linguagem. Atualmente, existem muitos compiladores que transformam uma especificação IDL em código para *proxies* e *stubs*, em uma linguagem de programação específica. Este é o caso do compilador MIDL, que acompanha a plataforma Windows e gera código para a linguagem C. Em resumo, portanto, a linguagem IDL é útil para descrever as interfaces e os dados compartilhados entre o cliente e um componente COM.

A linguagem Haskell suporta a criação de componentes COM, assim como a criação de clientes para componentes COM. Para isso, deve ser utilizada a ferramenta HDirect, apresentada na subseção a seguir.

5.3.2 A Ferramenta HDirect

HaskellDirect [21], ou simplesmente HDirect, é um compilador IDL para a linguagem Haskell. Em outras palavras, esta ferramenta permite que o programador Haskell interaja e reuse código escrito em outras linguagens, ao mesmo tempo em que pode expor o seu código Haskell para programadores de outras linguagens.

A principal tecnologia utilizada pelo HDirect é a IDL (*Interface Definition Language*). Conforme descrito na subseção anterior, uma especificação IDL define uma operação e seus parâmetros, que são implementados por uma linguagem de programação que possui um mapeamento para a sintaxe de IDL. No caso do HDirect, um mapeamento foi definido para permitir que a linguagem Haskell pudesse expor seus dados para IDL e também para que especificações IDL pudessem ser interpretadas por Haskell. Adicionalmente, o HDirect

permite que componentes COM sejam chamados a partir de Haskell e também que aplicações Haskell sejam encapsuladas como um componente COM [20]. Além disso, o HDirect suporta tanto o dialeto IIOP/CORBA, especificado pela OMG (*Object Management Group*), como dialeto OSF, especificado pelo DCE (*Distributed Computing Environment*) e que possui várias extensões do compilador IDL da Microsoft.

5.3.3 Estruturação do Ambiente de Desenvolvimento do Projeto[19]

Com o tempo, ficou clara a necessidade pela criação de um repositório centralizado para o projeto VHS, visto que estava havendo dificuldades para consistir as atualizações do *tarball* de Simon Marlow, realizadas por diferentes desenvolvedores. Deste modo, foi solicitada, pelo próprio Marlow, a criação de uma área específica para o projeto VHS no repositório oficial da Haskell.org. Entretanto, para o acesso com permissão de escrita de uma área de projeto do repositório da Haskell.org, um desenvolvedor necessita cadastrar uma conta no repositório, ter o conhecimento dos utilitários CVS e SSH e dominar a geração de chaves públicas e privadas para satisfazer as medidas de segurança demandadas pela Haskell.org. O aluno deste Trabalho de Graduação interagiu com os responsáveis pelo repositório da Haskell.org e documentou, no Plano de Ambiente e no Plano de Gerência de Configuração, o procedimento necessário para a configuração do CVS/SSH em uma máquina cliente e a utilização desses utilitários para o acesso e submissão de arquivos ao repositório, de modo a facilitar o trabalho dos demais desenvolvedores do projeto VHS-AM.

Em seguida, foi delegada ao aluno Mauro Araújo a configuração do ambiente de desenvolvimento do projeto, sendo o mesmo responsável pela instalação dos softwares necessários nas máquinas, como o Cygwin, VS.NET, AM, etc. A pedido do professor André Santos, o suporte do CIn liberou o laboratório C6 para a equipe do projeto, sendo o mesmo utilizado como ambiente de desenvolvimento juntamente com as máquinas pessoais de cada desenvolvedor. A configuração deste ambiente é apresentada na Tabela 4.

Infelizmente, devido à concorrência do laboratório C6 com alunos do mestrado e também a alguns incidentes isolados, como a formatação completa do C6 sem aviso prévio à equipe do projeto, algumas medidas adicionais foram tomadas: duas baias no antigo CESAR foram alocadas para o projeto, sendo uma ocupada por uma máquina do laboratório C6 e outra pela própria máquina que ficava na sala do professor André Santos.

---

[19] Um ambiente de desenvolvimento de projeto consiste em um conjunto de máquinas, com uma configuração de hardware e software específica, que será utilizado pelos desenvolvedores do projeto. Este conceito não deve ser confundido com ambientes integrados de desenvolvimento, ou IDEs.

Tabela 4 – Ambiente de desenvolvimento do projeto VHS-AM

Hardware	Software	Finalidade
Nome da máquina: c6c01 **Configuração:** Athlon XP 2000+ 786Mb **SO:** Windows 2000 Professional	VS.NET 2003 VSIP VSIP Extras Cygwin	Suportar desenvolvimento de VSPackages
Nome da máquina: andresantos **Configuração:** Pentium IV 2.4GHz 1GMB **SO:** Windows Server 2003 Standard Edition	AM Server	Servidor AM
Nome da máquina: c6c02 **Configuração:** Athlon XP 2000+ 786Mb **SO:** Windows 2000 Professional	AM Faculty Client	Simular o uso do Professor AM
Nome da máquina: c6c04 **Configuração:** Athlon XP 2000+ 786Mb **SO:** Windows 2000 Professional	AM Student Client	Simular o uso do Aluno AM
Nome da máquina: AW **Configuração:** Pentium3 800Mhz 256Mb **SO:** Windows XP Professional Edition	VS.NET 2003 VSIP VSIP Extras Cygwin	Máquina de desenvolvimento de awbf
Nome da máquina: peserfone **Configuração:** Athlon 1Ghz 368Mb **SO:** Windows XP Professional Edition	VS.NET 2003 VSIP VSIP Extras Cygwin	Máquina de desenvolvimento de mscla
Nome da máquina: menezes **Configuração:** Athlon 1.33Ghz 256Mb **SO:** Windows XP Professional Edition	VS.NET 2003 VSIP VSIP Extras Cygwin	Máquina de desenvolvimento de mmc3
Nome da máquina: alms **Configuração:** Pentium3 1Ghz 256Mb **SO:** Windows XP Professional Edition	VS.NET 2003 VSIP VSIP Extras Cygwin	Máquina de desenvolvimento de alms

A dificuldade no estabelecimento do ambiente de desenvolvimento do projeto foi uma lição aprendida, que levou o aluno deste Trabalho de Graduação a estabelecer, como ambiente de testes do projeto, máquinas mais acessíveis. Foram utilizadas, portanto, máquinas do Centro de Tecnologia XML do Recife [63] para este ambiente, ao qual o aluno possui acesso e é bastante grato. A configuração das máquinas neste ambiente é apresentada na Tabela 5.

Tabela 5 – Ambiente de testes do projeto VHS-AM

Hardware	Software	Finalidade
Nome da máquina: Faramir **Configuração:** Pentium 4 1,8Ghz 640MB **SO:** Windows XP Professional Edition	AM Server	Servidor AM
Nome da máquina: Boromir **Configuração:** Pentium 4 1,8Ghz 640MB **SO:** Windows XP Professional Edition	VS.NET 2003 com AM Faculty Client	Simular o uso do Professor AM
Nome da máquina: Krasimir **Configuração:** Pentium 4 1,8Ghz 640MB **SO:** Windows XP Professional Edition	VS.NET 2003 com AM Student Client	Simular o uso do Aluno AM
Nome da máquina: Denethor **Configuração:** Pentium 4 1,8Ghz 640MB **SO:** Windows XP Professional Edition	VS.NET 2003	Simular o uso do programador Haskell

5.3.4 Validações tecnológicas

Uma vez com o ambiente de desenvolvimento e testes estabelecidos, a equipe do projeto VHS-AM passou a executar validações tecnológicas com o objetivo de investigar mais profundamente as tecnologias a serem utilizadas no projeto. Ao aluno Marden Menezes foi delegada a tarefa de estudar o então recém-lançado VSIP Extras Beta, enquanto Mauro Araújo ficou responsável por verificar o funcionamento da ferramenta Assignment Manager. O aluno deste Trabalho de Graduação se responsabilizou por testar o MyC, exemplo de linguagem suportada pelo VS.NET que acompanha o VSIP. Em uma etapa posterior, o MyC foi testado em conjunto com o Assignment Manager, revelando, surpreendentemente, uma falha nesta validação tecnológica integrada: o AM não reconheceu como válido o tipo de projeto implementado no exemplo do MyC, colocando em questão se o AM seria capaz de suportar novos tipos de projeto criados a partir do VSIP. Este problema foi levantado no *newsgroup* do VSIP e diagnosticado como possível *bug* por alguns, mas não houve uma conclusão mais concreta sobre o assunto. Uma outra validação tecnológica, realizada em conjunto com Mauro Araújo, pôde tranqüilizar a equipe de projeto: a ferramenta AM reconheceu sem problemas o tipo de projeto implementado pelo SML.NET, revelando que o ocorrido com o MyC, provavelmente, deveria se tratar de um *bug*.

5.3.5 Implementação das interfaces do VSIP pelo projeto VHS

Após as validações tecnológicas efetuadas na etapa anterior, o único conhecimento ainda não dominado pela equipe do projeto VHS-AM consistia no procedimento utilizado pelo código original do projeto VHS para a implementação das interfaces do VSIP. Esse código foi,

portanto, estudado em mais detalhes e debatido em algumas discussões técnicas, lideradas pelo professor André Santos. Um resumo dessas discussões segue abaixo.

As interfaces do VSIP, incluindo as do *framework* do Babel, estão especificadas em IDL. Desse modo, é necessário utilizar um compilador IDL, como o HDirect, para converter as interfaces do VSIP e do Babel em código-fonte, através do qual possa ser iniciado o processo de implementação. Uma peculiaridade, entretanto, é que o arquivo que contém as interfaces do Babel, *babelservice.idl*, contém não apenas a interface *IBabelService* (a ser implementada pelo programador), como também outras interfaces a serem utilizadas pelo programador, cuja implementação já é realizada pelo *framework* do Babel. Desse modo, foi necessário dividir o arquivo *babelservice.idl* em outros dois arquivos:

- ***HaskellService.idl*:** contém a especificação, em IDL, da interface *IBabelService*, que deve ser implementada de modo a construir o suporte à edição de linguagem Haskell no VS.NET;
- ***BabelServiceLib.idl*:** contém o conjunto das demais interfaces especificadas no arquivo *babelservice.idl* (*IBabelPackage*, *IBabelProject*, *IparseSink*, *IColorSink*, etc.), cuja implementação já está pronta deve ser chamada pelo código que implementa a interface *IBabelService*.

A Figura 37 ilustra o processo completo utilizado para a implementação, em Haskell, das interfaces do Babel. Após a separação manual do arquivo *babelservice.idl*, conforme explicado acima, a ferramenta HDirect foi utilizada em cada um dos dois arquivos IDL resultantes dessa separação, com o objetivo de gerar arquivos contendo código Haskell. Entretanto, as opções de geração de código passadas ao HDirect foram diferentes em cada caso. No primeiro, referente ao arquivo *BabelServiceLib.idl*, a intenção foi gerar um arquivo em Haskell que provesse acesso ao *VSPackage* do Babel. No segundo, referente ao arquivo *HaskellService.idl*, a intenção foi gerar um esqueleto de implementação (*HaskellService.hs*) a ser preenchido pelo programador e que se comunica com o *framework* do Babel através do código gerado a partir do arquivo *BabelServiceLib.idl*. Para expor a implementação desse esqueleto como um componente COM, o HDirect gera, adicionalmente, um *proxy* que deve ser compilado em conjunto com o esqueleto implementado para a construção, por fim, do *VSPackage* a ser utilizado pelo VS.NET. Infelizmente, a geração de código pelo HDirect encontra-se com problemas (mais de 15 *bugs* foram identificados por Simon Marlow), sendo necessária a realização de ajustes manuais nos arquivos Haskell gerados pela ferramenta.

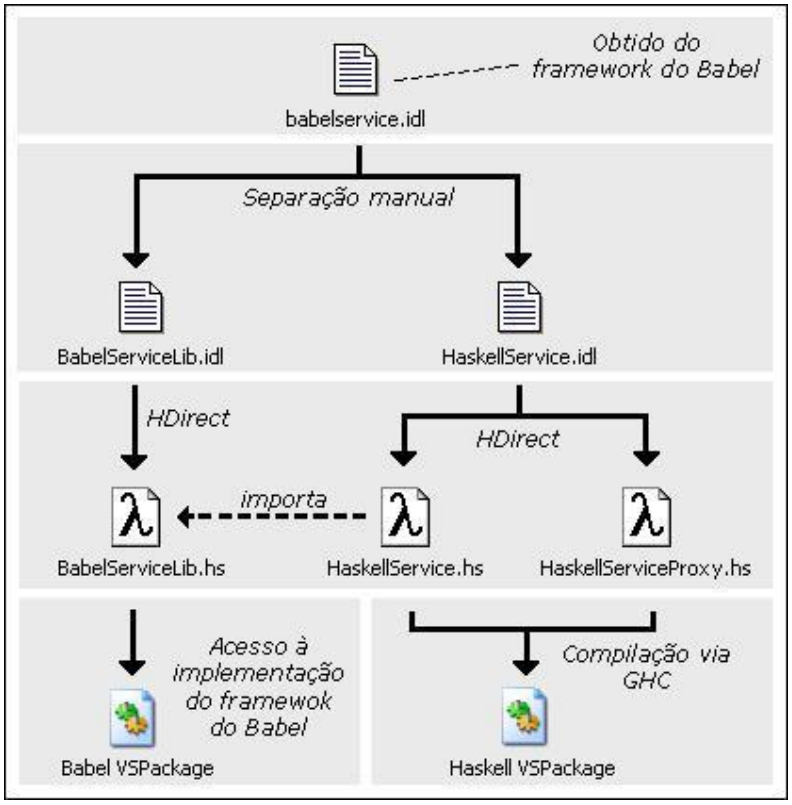

Figura 37 – Implementação, em Haskell, de suporte à linguagem com o Babel

Apesar da explicação acima estar focada em suporte à edição e utilização do Babel, o processo apresentado também se aplica à implementação de demais funcionalidades e interfaces do VSIP. Em resumo, sempre se faz necessária a aplicação do HDirect para a geração não só de esqueletos de implementação como também do código de acesso aos componentes já implementados e disponibilizados pelo VSIP.

5.4 Casos de Uso e RNFs Implementados: o Status Atual do Projeto VHS-AM

Enquanto a equipe do projeto VHS-AM estava focada na elaboração dos documentos da Fase de Planejamento (Documento de Especificação Funcional e o Plano de Projeto), além da realização das validações tecnológicas e análises necessárias para garantir o domínio das novas tecnologias e ferramentas a serem utilizadas, conforme apresentado na seção anterior, a equipe do projeto VHS original realizou alguns avanços. Simon Marlow conseguiu integrar o Haskell *VSPackage* (antigo *tarball*, agora no repositório da Haskell.org) com o compilador GHC, permitindo a exibição de mensagens de erro mais precisas, inclusive erros de tipo, na *tasklist* do VS.NET (Figura 38).

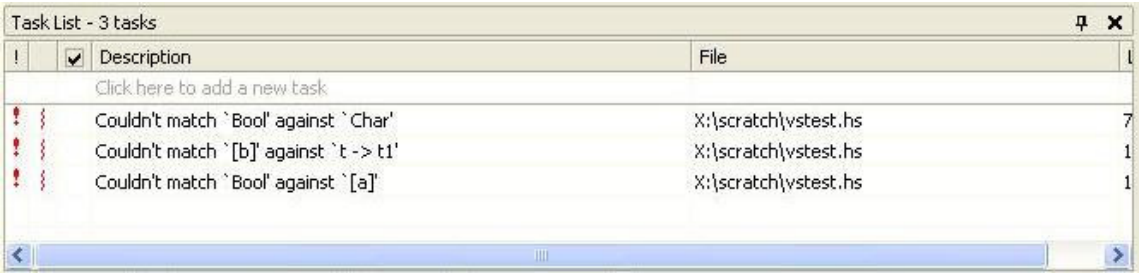

Figura 38 – Mensagens de erro do próprio GHC exibidas na *tasklist* do VS.NET

Enquanto isso, Krasimir Angelov iniciou a implementação de um suporte básico a projetos Haskell no VS.NET, a partir da implementação das interfaces do VSIP (pois não existe um *framework* como o Babel para auxiliar o desenvolvimento de suporte a projetos, ao contrário do suporte à edição).

Ao iniciar a Fase de Desenvolvimento, portanto, este foi o cenário encontrado pela equipe do projeto VHS-AM: *syntax coloring* e mensagens de erro na *tasklist*, advindas do GHC, já estavam funcionando, enquanto um suporte básico a projetos estava sendo iniciado. De modo a visualizar os avanços relativos ao suporte à edição, entretanto, a equipe precisou passar passou por um complexo processo de configuração do ambiente de projeto: a compilação do GHC de modo que seu *runtime system* suportasse múltiplas *threads*, funcionalidade recém-criada pelos desenvolvedores da Haskell.org. Isso foi necessário porque a versão mais recente do Haskell *VSPackage* era chamada pelo Babel através de duas *threads* simultâneas: uma principal e outra específica para a invocação do *parser*. De certa forma, a utilização desse *threaded runtime system* pela equipe do projeto VHS-AM serviu como um teste *beta* para a sua incorporação definitiva a versões posteriores do GHC.

De acordo com a priorização dos casos de uso discutida no Capítulo 3, o primeiro *release* interno do projeto VHS-AM deveria suportar uma integração mínima de Haskell com a ferramenta Assignment Manager, permitindo a publicação e resolução de exercícios nessa linguagem. Foi identificado que, de modo a viabilizar tal integração, era necessário obter um suporte a projetos Haskell no VS.NET que permitisse a execução das seguintes tarefas:

- Criação de um novo projeto Haskell, contendo pelo menos um módulo Haskell;
- Edição básica do módulo contido no projeto;
- Compilação do projeto;
- Execução do projeto.

Infelizmente, o suporte a projeto até então desenvolvido por Krasimir Angelov atendia apenas ao primeiro dos requisitos acima, sendo necessário à equipe do VHS-AM a procura por uma outra alternativa, de modo a concluir o primeiro release *interno* dentro do cronograma e prestar contas com a *Microsoft Shared Source Initiative*. Três alternativas, portanto, foram abordadas:

- Marden Menezes verificaria a possibilidade de alterar o suporte a projeto do MyC, exemplo em C++ que acompanha o VSIP, para suportar projetos Haskell;
- Mauro Araújo analisaria o esforço necessário para fazer o suporte a projeto de Krasimir Angelov atender a todos os requisitos necessários para o Assignment Manager;
- O aluno deste Trabalho de Graduação analisaria o suporte a projetos Haskell desenvolvido no passado por Simon Marlow para o Visual Studio .NET 2002, mas que até então estava descontinuado e não funcionava para a versão 2003 da IDE[20].

Após alguns dias de análise, a terceira alternativa mostrou-se a mais viável, sendo o antigo suporte a projetos Haskell migrado com sucesso para o Visual Studio .NET 2003. Tal alternativa, baseada originalmente no suporte a projetos para a linguagem SML.NET, consistia em utilizar VS.NET *wizards* para criar um novo *template* de projeto do tipo *Visual C++ Makefile*, que permite a especificação do processo de compilação em mais detalhes pelo programador (ou seja, o GHC poderia ser utilizado para compilar este tipo de projeto). Deste modo, o esforço para a migração envolveu:

- Manipulação das entradas do registro necessárias para o novo *wizard*;
- Elaboração dos arquivos de configuração .vsdir e .vsz utilizados pelo novo *wizard*;
- Criação de uma dll de recursos (*localized resource*) contendo ícones, textos e outros recursos a serem utilizados pelo novo *wizard*;
- Adequação do arquivo JavaScript acionado ao fim do *wizard* para a criação do projeto Haskell, especificação do GHC como compilador e cópia dos arquivos iniciais necessários.

O aluno Marden Menezes foi alocado para a criação de um instalador que implantasse esse suporte a projetos Haskell no VS.NET. Utilizando o próprio VS.NET, o instalador foi criado com a tecnologia *Microsoft Installer*, gerando um arquivo .MSI que conduz intuitivamente o usuário do início ao fim da instalação. Um *screenshot* desse instalador em ação é apresentado na Figura 39.

Após a conclusão bem-sucedida dos testes e a elaboração de *release notes*[21], pelo aluno deste Trabalho de Graduação, o primeiro *release* interno estava concluído. Com o início do desenvolvimento do segundo *release* interno, o foco da equipe passou do AM para o próprio VS.NET.

---

[20] Este suporte a projeto havia sido abandonado porque a implementação sugerida por Krasimir Angelov era mais extensível, permitindo a adição de mais funcionalidades no futuro.

[21] Segundo a metodologia Pro.NET, os *release notes* de uma aplicação devem ser elaborados ao fim de cada *release* (interno ou não), apresentando suas funcionalidades, defeitos conhecidos, requisitos de *hardware* e *software*, instruções de instalação e informações de contato.

Figura 39 – *Screenshot* do Instalador do suporte a projetos Haskell para o VS.NET

Com a conclusão da implementação do suporte a projetos no primeiro *release* interno, foi aberto o caminho para a implementação de casos de uso que dependiam desse suporte a projetos, como a adição de um novo módulo Haskell a um projeto (UC05) e a implementação de um *module viewer* para Haskell (UC06), não havendo, a princípio, a necessidade de esperar a conclusão dos trabalhos de Krasimir Angelov.

A implementação da adição de um novo módulo a um projeto Haskell foi realizada com sucesso pelo aluno deste Trabalho de Graduação, através de um "*wizard* de item" (ver Capítulo 4). A implementação do *module viewer*, entretanto, mostrou-se um desafio muito mais complexo. Após uma semana e meia de análises, chegou-se à conclusão de que não seria possível implementar o *module viewer* sem a existência de um suporte a projeto implementado através do VSIP, trabalho que estava sendo realizado por Krasimir Angelov. Deste modo, decidiu-se por mover este caso de uso para o terceiro *release* interno, trocando-o pelo o caso de uso UC01E: "Desenvolver código Haskell com suporte à edição: *method tip*". Assim, o foco principal do segundo *release* interno consistiu em implementar demais funcionalidades de suporte à edição através do Babel.

A implementação de *brace matching* (casamento entre pares de chaves/parênteses) foi parcialmente bem-sucedida. Através do Babel, o professor André Santos conseguiu implementar o redirecionamento do cursor entre pares de chaves/parênteses, através da combinação de teclas **CTRL** e **]** (padrão do VS.NET). Adicionalmente, é exibida com sucesso, na *satus bar* do VS.NET, a linha de código que contém o par correspondente de um parênte-se/chave, conforme mostra a Figura 40. Entretanto, por razões ainda desconhecidas, não foi possível ainda visualizar o efeito *highlight braces*, em que o par de parênteses/chaves fica em negrito quando o cursor passa sobre um deles.

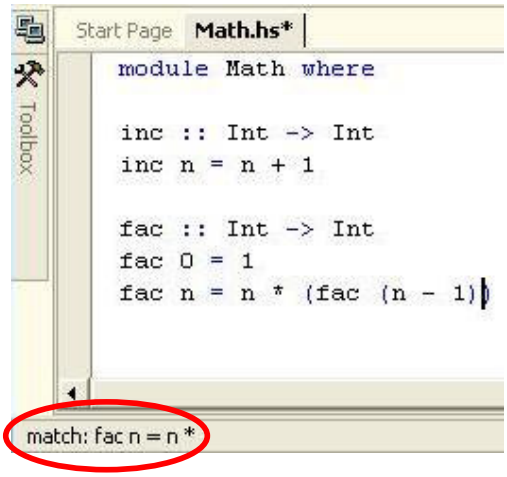

Figura 40 – *Brace matching* para Haskell no VS.NET

Similarmente, avanços parciais foram realizados em relação às outras funcionalidades relativas ao suporte à edição baseado no Babel. É possível verificar com sucesso o surgimento de uma *popup list* cujos itens dependem do escopo no qual está o cursor, implementado pelo aluno deste Trabalho de Graduação, conforme mostra a Figura 41.

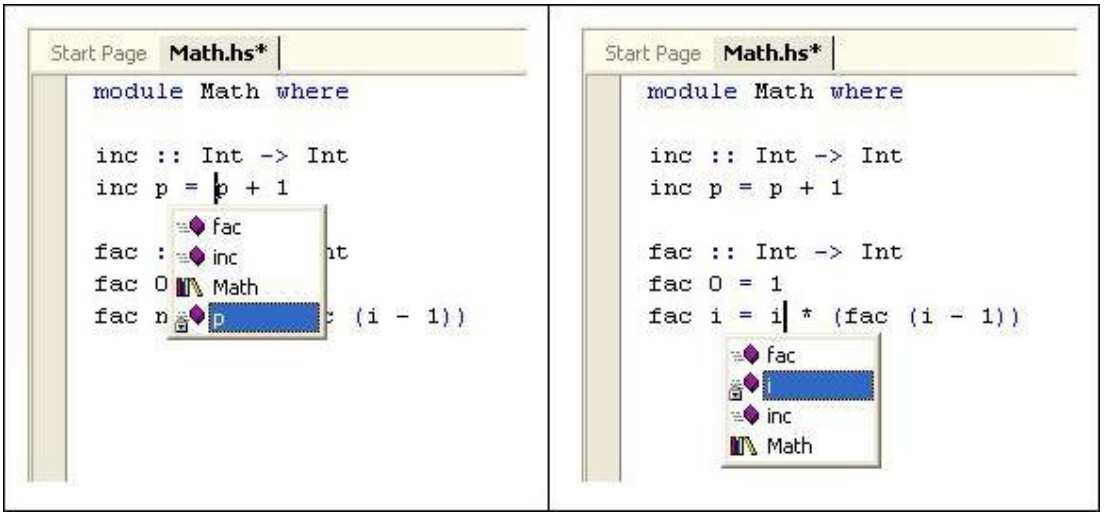

Figura 41 – *Popup list* dependente do escopo no qual está o cursor

Entretanto, ainda não se conseguiu visualizar funcionalidades como *quick info* e *method tip*. Infelizmente, a documentação do VSIP específica sobre o Babel parece estar desatualizada, o que dificulta a busca pela causa dos problemas de suporte à edição ainda não-resolvidos. Atualmente, de modo a isolar esses problemas, o aluno deste Trabalho de Graduação está realizando depurações não apenas no *parser* Haskell utilizado pelo Haskell *VSPackage*, como também no próprio *VSPackage* do Babel, cujo código-fonte acompanha a versão Extras do VSIP.

Paralelamente, o restante da equipe do projeto VHS-AM avança em outras áreas. Marden Menezes verificou com sucesso o procedimento para a obtenção de uma *Package Load Key*, necessária para registrar um *VSPackage* em versões não experimentais do VS.NET. Atualmente, Menezes está iniciando a implementação do caso de uso referente à consultas ao interpretador GHCi. Enquanto isso, Mauro Araújo foi alocado para integrar o Haskell *VSPackage* com o GHC, permitindo que a equipe verifique os avanços já obtidos por Simon Marlow e possa estendê-los[22]. Tal tarefa implica em compilar o código-fonte do GHC com o objetivo de gerar uma dll, que expõe funcionalidades do próprio GHC (como seu verificador de tipos) que podem ser utilizadas no Haskell *VSPackage*.

[22] Atualmente, a equipe do projeto VHS-AM utiliza um *parser* "*standalone*" e não o *parser* do próprio GHC. Este *parser*, também feito em Haskell, foi disponibilizado com a publicação do *tarball* de Simon Marlow.

6. A METODOLOGIA PRO.NET NO PROJETO VHS-AM

Todo projeto que integra diferentes tecnologias envolve riscos. Além disso, a interação entre projetos de objetivos similares é uma tarefa que demanda disciplina e coordenação. Por fim, a gerência de tempo, escopo, recursos e dos próprios riscos implica na necessidade de um processo prático, sistemático e bem-definido para a condução de projetos de desenvolvimento de software. Por esses e demais motivos, a aluno deste Trabalho de Graduação sugeriu que ao projeto VHS-AM fosse aplicada a metodologia Pro.NET, desenvolvida pelo Centro de Tecnologia XML do Recife e cuja criação teve a participação do próprio aluno. Este capítulo apresenta a metodologia e ao mesmo tempo discute como ela vem sendo aplicada ao projeto. Boa parte das informações aqui descritas resultou do trabalho desenvolvido no Centro de Tecnologia XML, tendo sido reproduzidas neste relatório e também e apresentadas aos demais membros do projeto VHS-AM, pelo aluno deste Trabalho de Graduação, com a anuência do próprio Centro de Tecnologia XML.

6.1 O Centro de Tecnologia XML e a Criação da Pro.NET

O Centro de Tecnologia XML do Recife é uma iniciativa que faz parte de um projeto maior lançado pela Microsoft, em 2002, para a criação de 20 centros de excelência tecnológica no país. Em Recife, o Centro tem como gestores o Centro de Estudos e Sistemas Avançados do Recife (CESAR) e a Qualiti Software Processes, sendo financiado pela Microsoft Brasil, HP Brasil e a Financiadora de Estudos e Pesquisas (FINEP). Como demais parceiros, o Centro de Tecnologia XML do Recife conta com o Centro de Informática (CIn) da UFPE, o complexo tecnológico Porto Digital e a Empresa de Fomento à Informática do Estado de Pernambuco (FISEPE).

A FINEP está apoiando o Centro XML através de um projeto intitulado Processos de Desenvolvimento de Software para a Plataforma .NET (Pro.NET), aprovado em dezembro de 2002. O proponente deste projeto é o Centro de Estudos Avançados do Recife (CESAR), seus executores são o próprio CESAR e o CIn e os intervenientes são a Microsoft Informática LTDA e o núcleo de gestão do Porto Digital.

A partir deste projeto, portanto, surgiu a metodologia Pro.NET, essencialmente baseada no *Microsoft Solutions Framework* (MSF) e no *Rational Unified Process* (RUP). Adicionalmente, contribuíram para a metodologia o *Project Management Body of Knowledge* (PMBOK) e o *Extreme Programming* (XP). Apesar de ser focada para o desenvolvimento de software para a plataforma .NET, a Pro.NET permite sua utilização em um nível maior de abstração, sendo este o caso do projeto VHS-AM.

6.2 Estruturação da Pro.NET

Os principais elementos que compõem a estrutura da metodologia Pro.NET são o seu Modelo de Equipe e o Modelo de Processo. Ambos os modelos serão detalhados nas subseções a seguir.

6.2.1 O Modelo de Equipe

O Modelo de Equipe da Pro.NET, que descreve como estruturar a equipe e suas responsabilidades para atingir sucesso do projeto, tem como base seis princípios fundamentais:

- **Estabelecer uma visão compartilhada do projeto:** essa visão permite esclarecer os objetivos do projeto e trazem à tona conflitos e asserções erradas para que os mesmos possam ser resolvidos. Uma visão compartilhada é essencial ao sucesso de um projeto.
- **Focar no valor agregado ao negócio do cliente:** a equipe deve estar atenta ao que é realmente importante para o negócio do cliente. Desde o início, deve estar claro que a tecnologia é utilizada como meio, e não como foco.
- **Permanecer ágil e esperar mudanças:** todos os membros da equipe devem estar cientes de que mudanças ocorrerão e modificarão seu trabalho. Deve ser estimulada a prática de revisões e sugestões no trabalho realizado por outros membros, assim como o estabelecimento de medidas de rastreamento para garantir eficiência na implantação de mudanças.
- **Incentivar comunicação aberta:** a comunicação entre os membros da equipe deve ser estimulada e coordenada. Isso reduz os enganos provenientes da falta de informação. Se precisarem existir informações secretas, a equipe deve estar ciente que existe o sigilo e que ele contribui para o sucesso do projeto.
- **Compartilhar responsabilidade:** cada membro da equipe é ciente das suas atribuições e divide a responsabilidade pelo sucesso do projeto.
- **Dar a liberdade necessária e confiar nos membros da equipe:** significa entregar aos membros da equipe a autoridade e os recursos necessários para preencher as responsabilidades associadas com seus papéis.

Os papéis desempenhados pelos membros de um projeto, segundo o Modelo de Equipe da Pro.NET, não são hierarquizados, isto é, cada papel da equipe é único, importante e igualmente valioso. A ausência de uma hierarquia, contudo, não implica na ausência de uma coordenação do projeto. Esta coordenação é executada pelo papel *Program Management*[23]. Este e os demais papéis da Pro.NET são apresentados na Figura 42, estando as responsabilidades de cada papel descritas na Tabela 6.

[23] No projeto VHS-AM, foi decidido preservar os nomes dos papéis em inglês, pois a tradução utilizada pela Pro.NET será modificada em um futuro próximo.

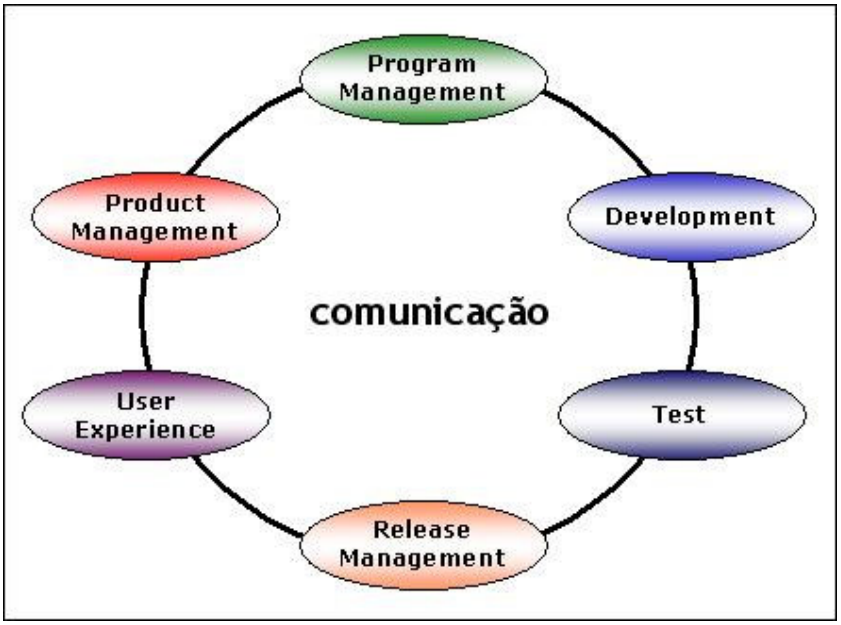

Figura 42 – Modelo de Time da Pro.NET (não traduzido)

Tabela 6 – Responsabilidades dos papéis da Pro.NET

Papel	Responsabilidades
Program Management	Garante a condução do projeto dentro de suas restrições, como recursos e cronograma. Facilita a comunicação da equipe, gerencia todo processo de riscos, reporta o status do projeto e mantém seus principais artefatos, como o Plano de Projeto e o Documento de Especificação Funcional.
Product Management	Age como advogado do cliente, representado seus interesses. É um dos principais responsáveis pela gerência de decisões de escopo *vs.* cronograma *vs.* recursos.
Development	Constrói e supervisiona a implementação de casos de uso e prepara aplicação para implantação.
Test	Define e conduz o processo de testes no projeto, garantindo que todos os defeitos estão identificados.
User Experience	Age como advogado do usuário final para a equipe, representado seus interesses. Especifica funcionalidades e mecanismos de educação dos usuários, além de gerenciar decisões de usabilidade e aumento da produtividade do usuário
Release Management	Gerencia a implantação da solução, oferece suporte técnico à equipe e gerencia os ambientes do projeto

No caso do projeto VHS-AM, a divisão da equipe entre os papéis da Pro.NET se deu conforme mostra a Tabela 7. É importante observar que, como a equipe é composta por apenas 5 membros, muitas vezes mais de um papel foi alocado por membro. Após a definição dos papéis, os integrantes do projeto foram orientados em relação a suas funções e atividades que iriam desempenhar no projeto[24], sendo definida, adicionalmente uma freqüência semanal de reuniões, para acompanhar a evolução das atividades da equipe.

Tabela 7 – Divisão da equipe em papéis no projeto VHS-AM

Membro	Papéis
André Santos	• *Product Management* • *Test*
Paulo Borba	• *User Experience*
André Furtado	• *Development* • *Program Management* • *Test*
Mauro Araújo	• *Development* • *Release Management*
Marden Mezenes	• *Development* • *User Experience*

6.2.2 O Modelo de Processo

O Modelo de Processo corresponde à estruturação das atividades a serem realizadas durante todo o desenvolvimento da aplicação. Este modelo divide o projeto em várias iterações, cada uma contendo 5 fases distintas conforme apresentado na Figura 43: Visão, Planejamento, Desenvolvimento, Estabilização e Implantação. Como pode ser observado na própria Figura 43, o fim de cada fase é caracterizado por um marco, ou *milestone*.

A **Fase de Visão** é a primeira da iteração, podendo ser considerada um pré-planejamento do projeto. Neste momento, a equipe do VHS-AM identificou a motivação para o projeto, seus objetivos, produtos finais e restrições, elaborando também um macro-cronograma. O aluno deste Trabalho de Graduação consistiu as informações levantadas em um Documento de Visão e Escopo, aprovado em conjunto com o restante da equipe. O documento foi elaborado em inglês, de modo que também pudesse ser compartilhado com os membros do projeto VHS. O *vision statement*, que resume o sentido do projeto em uma única frase, foi definido como: "*To support a productive use of Haskell through VS.NET extension and AM integration, spreading around the utilization of the language and expanding its application boundaries*."

[24] O professor Paulo Borba participará da homologação final do projeto, validando-o para o usuário final.

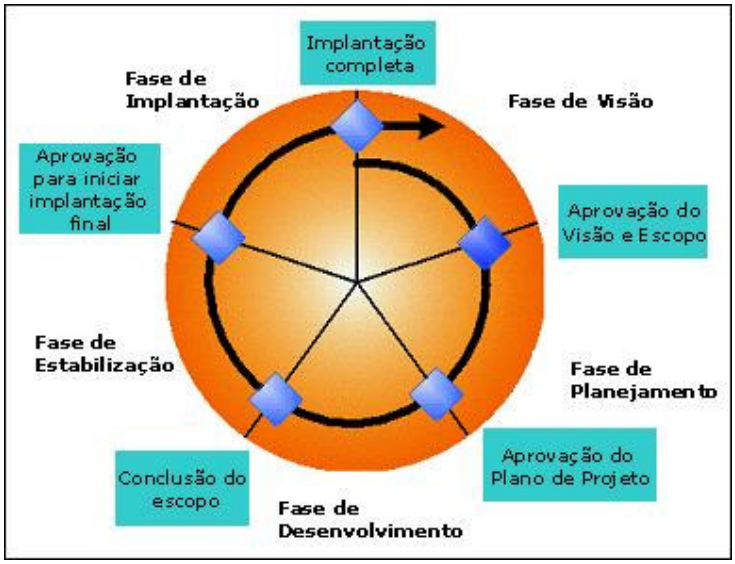

Figura 43 – Modelo de Processo da Pro.NET

Ainda na Fase de Visão, o aluno deste Trabalho de Graduação elaborou o Glossário inicial do projeto e desenvolveu uma Lista de Riscos. Esta lista, baseada em informações de toda a equipe, permitiu gerenciar o projeto em termos de suas principais deficiências. Uma "fotografia" da Lista de Riscos, ao fim da Fase de Visão, é exibida na Tabela 8.

Por fim, uma identificação dos *stakeholders*, isto é, dos indivíduos afetados de alguma maneira pelo projeto e a aplicação a ser desenvolvida, também foi realizada nesta fase, sendo apresentada abaixo:

- **Programadores Haskell:** usuários finais da aplicação. Serão beneficiados diretamente pelo projeto, pois irão dispor de um novo mecanismo para aumentar a produtividade da etapa de implementação.
- **Microsoft Corporation:** cliente do projeto VHS-AM. Através do programa *Microsoft Shared Source Initiative*, a empresa está investindo recursos para a realização projeto e espera um produto final compatível com a proposta submetida ao programa.
- **Equipes paralelas de desenvolvimento:** indivíduos externos ao projeto (Simon Marlow, Krasimir Angelov, etc.) que compartilham os mesmos objetivos de extensão do VS.NET para suportar Haskell. Deve haver uma interação e coordenação entre as equipes para permitir uma paralelização dos trabalhos e, conseqüentemente, um produto final mais completo.
- **Usuários do Assignment Manager e comunidade acadêmica em geral:** professores e alunos, beneficiados pela existência de uma estrutura para o ensino e aprendizado da linguagem Haskell.
- **Desenvolvedores do VSIP, VS.NET e Assignment Manager:** serão beneficiados pela base de conhecimento a ser gerada com o projeto VHS-AM, permitindo que os mesmos tenham acesso a um número maior de lições aprendidas.

Tabela 8 – Lista de Riscos do projeto VHS-AM ao fim da Fase de Visão

Risco	Conseqüência	Mitigação	Contingência
Não há documentação rica sobre o modelo de extensão/automação do VS.NET	Maior possibilidade de defeitos no código; curva de aprendizado maior	Leitura do manual do VSIP; procura por novidades na Web	Elaboração de um mini-workshop para estudo mais profundo da tecnologia; pedir auxílio externo
A equipe não é experiente em COM (combinado ou não com Haskell)	A curva de aprendizado pode ser maior	Estudo da tecnologia; elaboração de exemplos básicos, manutenção de uma base de conhecimento	Elaboração de um mini-workshop para estudo mais profundo da tecnologia; pedir auxílio externo (Sigbjorn, alguém experiente no próprio CIn, etc.)
O *tarball* de Simon não está estável nem foi compreendido	Inexistência de um baseline concreto para a implementação da aplicação	Estudo mais profundo do *tarball*, possivelmente documentando-o	Realização de um *code review* entre os membros para discussão do código; pedir auxílio mais intenso de Simon Marlow e à lista VisualHaskell
A equipe desconhece a utilização do Assignment Manager com novas linguagens estendidas para o Visual Studio .NET.	Problemas de integração entre a linguagem e o Assignment Manager podem ocorrer	Elaboração de protótipos iniciais para validação	Procurar por suporte externo (*newsgroups*, consultores da MS, etc.); alocar um membro da equipe exclusivamente para a solução do problema
Novas versões do VSIP estão sendo lançadas	A equipe pode estar usando uma tecnologia obsoleta; a curva de aprendizado pode ser maior	Estudo das novas versões; elaboração de exemplos básicos, manutenção de uma base de conhecimento	Reunião para identificar requisitos já implementados passíveis de *refactoring* e requisitos não implementados que sofrerão alteração da maneira como serão implementados
Inexistência de um local centralizado de desenvolvimento para a equipe	Trabalho realizado apenas nos computadores pessoais dos membros da equipe; pobreza na visualização dos resultados; validação menos real da solução em vários ambientes	Contactar o suporte do CIn para liberação de máquinas para o projeto; liberar acesso remoto à maquina da sala de alms	Intensificar a solicitação da liberação de máquinas ao suporte; reunião para discussão de novas possibilidades
Diferentes projetos (VHS e VHS-AM) compartilham objetivos	Algum trabalho pode ser desperdiçado	Divulgação do escopo do projeto VHS-AM na lista do VHS; coordenação colaborativa de atividades	Discussão clara e aberta na lista VisualHaskell sobre divisão de responsabilidades e tarefas; inclusão das responsabilidades das diferentes equipes no *site*
A maior parte da equipe desconhece a metodologia a ser utilizada	O projeto pode ser conduzido de maneira informal demais, prejudicando sua qualidade	Explicação da metodologia; alocação de tempo para estudo da mesma e acompanhamento para eventuais dúvidas	Reuniões para explicação do que está indo errado e apresentação do processo correto da metodologia
A equipe não possui um domínio avançado da linguagem Haskell	O produto final pode não ser aplicável à realidade dos programadores Haskell	Estudo avançado da linguagem; estimular *feedback* contínuo e proativo	Priorizar atividades de estudo da linguagem e análise de *feedback*
Não há previsão para a integração do VHS (de Simon) com o *parser* do GHC	Algumas funcionalidades não poderão ser oferecidas pelo VHS-AM	Acompanhar os avanços de Simon Marlow	Utilizar o *parser* atual, mais limitado

Durante a **Fase de Planejamento**, o aluno desenvolveu o Documento de Especificação Funcional do projeto (também em inglês), que detalha seus casos de uso e requisitos não-funcionais (Capítulo 3), e também o Plano de Projeto, que descreve as abordagens a serem seguidas para a condução do projeto em relação às diferentes áreas de conhecimento (Implementação, Testes, Implantação, etc.) Ambos os documentos foram discutidos e validados com toda a equipe, que também esteve envolvida na elaboração de validações tecnológicas para o melhor entendimento dos conceitos, tecnologias e ferramentas necessárias ao projeto. É importante destacar que nesta fase os principais documentos as serem produzidos no projeto foram registrados no Plano de Qualidade (que faz parte do Plano de Projeto), conforme mostra a Tabela 9.

Tabela 9 – Lista de Artefatos do projeto VHS-AM

Artefato	Idioma	Fase em que será desenvolvido	Responsável
Visão e Escopo	Inglês	Visão	André Santos
Atas de Reunião	Português	Todas	André Furtado
Glossário	Português	Todas	André Furtado
Validação de Produtos e Tecnologias	Português	Planejamento e Desenvolvimento	Marden Menezes
Plano de Projeto	Português	Planejamento	André Furtado
Especificação Funcional	Inglês	Planejamento	André Furtado
Planilha de Riscos	Português	Todas	André Furtado
Projeto de Testes	Português	Desenvolvimento	André Furtado
Release Notes	Inglês	Desenvolvimento e Estabilização	Mauro Araújo
Guia de Instalação	Inglês	Estabilização	Marden Menezes
Base de conhecimento externa	Inglês	Implantação	André Furtado
Relatório de Fechamento do Projeto	Inglês	Implantação	André Furtado

A **Fase de Desenvolvimento**, dividida em três *releases* internos, é a atual fase na qual se encontra o projeto VHS-AM. Além de alocar os recursos do projeto na implementação dos casos de uso e requisitos não-funcionais, com a orientação e aprovação do professor André Santos, o próprio aluno deste Trabalho de Graduação participa da implementação dos mesmos, dividindo-se entre tarefas gerenciais e de desenvolvimento. Esta fase será finalizada quando o escopo total da aplicação for construído, embora ainda possam existir defeitos.

Na **Fase de Estabilização**, os defeitos ainda existentes da aplicação serão corrigidos. A equipe estará focada em testar, priorizar e corrigir os defeitos encontrados, gerando novos *releases* internos, e em preparar a solução para a implantação no ambiente de produção. Um dos *releases* internos gerados, devidamente testado e aprovado, será utilizado para implantação final da solução e marcará o final desta fase.

A última fase de uma iteração é a **Fase de Implantação**. É nela que a implantação final será realizada e o conhecimento adquirido no projeto será capturado e documentado. No projeto VHS-AM, esta implantação será bastante breve, compreendendo apenas três atividades:

- O código final modificado do Assignment Manager será disponibilizado no *site* de seu *Project Space* (*MSDN Academic Alliance*), sendo também publicado no *site* do projeto VHS-AM.
- O código final dos *VSPackages* desenvolvidos será disponibilizado no repositório da Haskell.org. As dlls desenvolvidas (ou instaladores MSI das mesmas) estarão disponibilizadas no *site* dos projetos VHS e VHS-AM, assim como um manual de instalação e um documento de lições aprendidas.
- Um e-mail formal deverá enviado à lista haskell@haskell.org, alertando sobre a conclusão bem sucedida do projeto.

A implantação final da solução é o último marco da iteração corrente e caracteriza o início da próxima iteração. O projeto VHS-AM prevê a realização de uma segunda (e rápida) iteração, caso o cronograma permita. Entretanto, esta segunda iteração não foi detalhada.

Além da estruturação do Modelo de Processo em um aspecto temporal, representado pela sucessão das fases apresentadas acima, existe uma estruturação atemporal, que organiza as atividades do projeto não em relação ao tempo (como as fases), e sim de acordo com a área de conhecimento. A metodologia Pro.NET divide as áreas de conhecimento, também chamadas de disciplinas, em dois grupos: disciplinas de processo e disciplinas de suporte.

As disciplinas de processo estão relacionadas ao trabalho de desenvolver a solução. São elas: Requisitos, Análise & Projeto, Implementação, Testes e Implantação. A disciplina de **Requisitos** busca entender a estrutura e a dinâmica do cliente, seus problemas correntes e identificar melhorias de processos, de modo a garantir que os clientes, os usuários finais e a equipe de desenvolvimento tenham um entendimento comum da solução a ser desenvolvida e de como esta solução atende aos objetivos da organização. O uso de protótipos é estimulado para aprimorar a especificação. **Análise & Projeto**, por sua vez, é a disciplina que possibilita a transformação dos requisitos em um projeto da aplicação, sugerindo uma arquitetura em camadas robusta e uma solução que atenda aos requisitos de implantação.

A disciplina de **Implementação** produz o código em várias camadas, implementa as funções e estruturas de dados, testa o código desenvolvido como unidade e realiza inspeção nos artefatos criados. A disciplina de **Testes** verifica a integração de partes de código implementadas separadamente, se todos os requisitos foram corretamente implementados e identifica e garante que defeitos serão resolvidos antes da implantação da solução. A disciplina de **Implantação**, por fim, disponibiliza versões da solução para os usuários e realiza treinamentos.

As disciplinas de suporte são formadas por um conjunto de atividades de apoio que possibilitam a execução das disciplinas de processo. São elas: Planejamento & Gerenciamento, Riscos e Ambiente & Gerencia de Configuração.

A disciplina de **Planejamento & Gerenciamento** abrange os principais aspectos do planejamento e gerenciamento de projetos em um conjunto de atividades bem definidas. O planejamento agrupa os vários planos criados (sendo alguns deles em outras disciplinas) em um Plano de Projeto integrado, considerando uma data fixa para a entrega da solução e o uso de *buffer time* para acomodar futuros problemas ao longo do desenvolvimento. O gerenciamento do projeto deve assegurar que as atividades programadas estão sendo executadas, gerenciar mudanças e garantir a qualidade dos artefatos produzidos e dos processos executados. Ao final do projeto, devem ser capturadas as lições aprendidas. A disciplina de **Riscos**, por sua vez, descreve uma maneira pró-ativa de lidar com incerteza e a considera nas decisões no ciclo de vida do projeto. A disciplina inclui identificação, priorização, planejamento, monitoração e controle dos riscos. Ao longo do projeto, é essencial a captura de lições aprendidas com os riscos. A disciplina **Ambiente & Gerencia de Configuração**, por fim, descreve como tratar a evolução dos artefatos produzidos, a geração de *releases* e *builds* de uma aplicação sendo desenvolvida e o controle das solicitações de mudanças na solução. Uma prática sugerida é a integração contínua do código da aplicação. Outro objetivo é planejar e gerenciar o ciclo de vida dos ambientes e dos recursos de *hardware* e *software* do projeto durante o progresso da solução.

A Figura 44 mostra como as disciplinas são executadas durante uma iteração. As disciplinas de suporte, que aparecem ao centro, são executadas de forma contínua durante toda a iteração.

O conceito de disciplinas possui grande impacto no desenvolvimento do projeto, visto que os artefatos do mesmo são catalogados por disciplina. Como exemplo, a Figura 45 mostra a estrutura de repositório criada para o projeto VHS-AM, durante o fim da Fase de Visão. Juntamente com os diretórios de cada disciplina, existe um diretório específico para a base de conhecimento, enriquecida durante todo o projeto. Normalmente, essa base se localiza em um repositório central da organização, não sendo parte de um projeto específico.

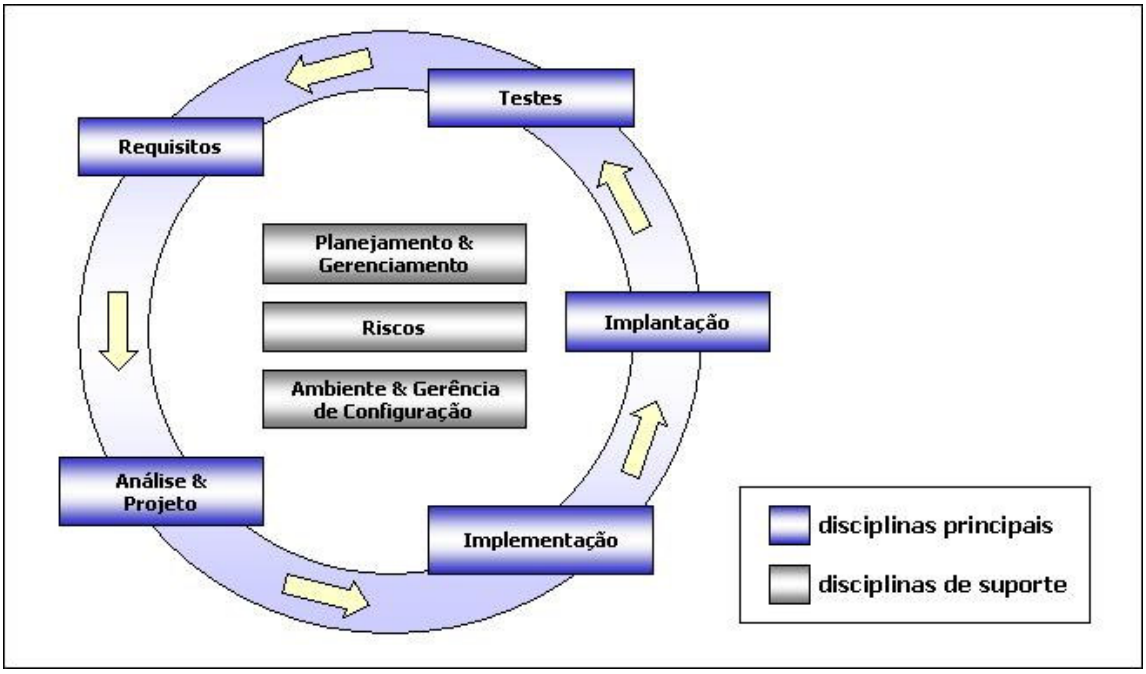

Figura 44 – Relacionamento entre disciplinas na Pro.NET

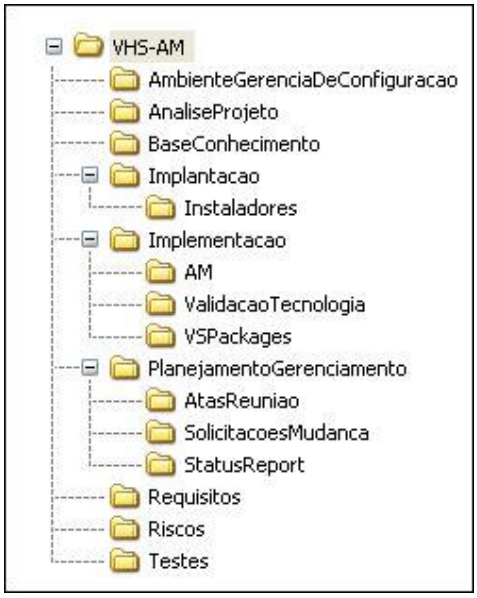

Figura 45 – Repositório do projeto VHS-AM

A Figura 46 relaciona o aspecto atemporal e temporal da Pro.NET. As fases estão na primeira linha e as disciplinas na primeira coluna. As células centrais representam quais são as macro-atividades a serem realizadas naquelas fases e disciplinas. Macro-atividades contêm um conjunto de atividades que são executadas conjuntamente para a realização de um objetivo. Elas fornecem um nível mais elevado de abstração na descrição do trabalho a ser

realizado. Para cada macro-atividade, a metodologia descreve o fluxo de execução de suas atividades. A Figura 46 apresenta macro-atividades associadas, cada uma, a uma disciplina e várias fases. A barra de cada macro-atividade representa o período do projeto em que existirá esforço para executá-la.

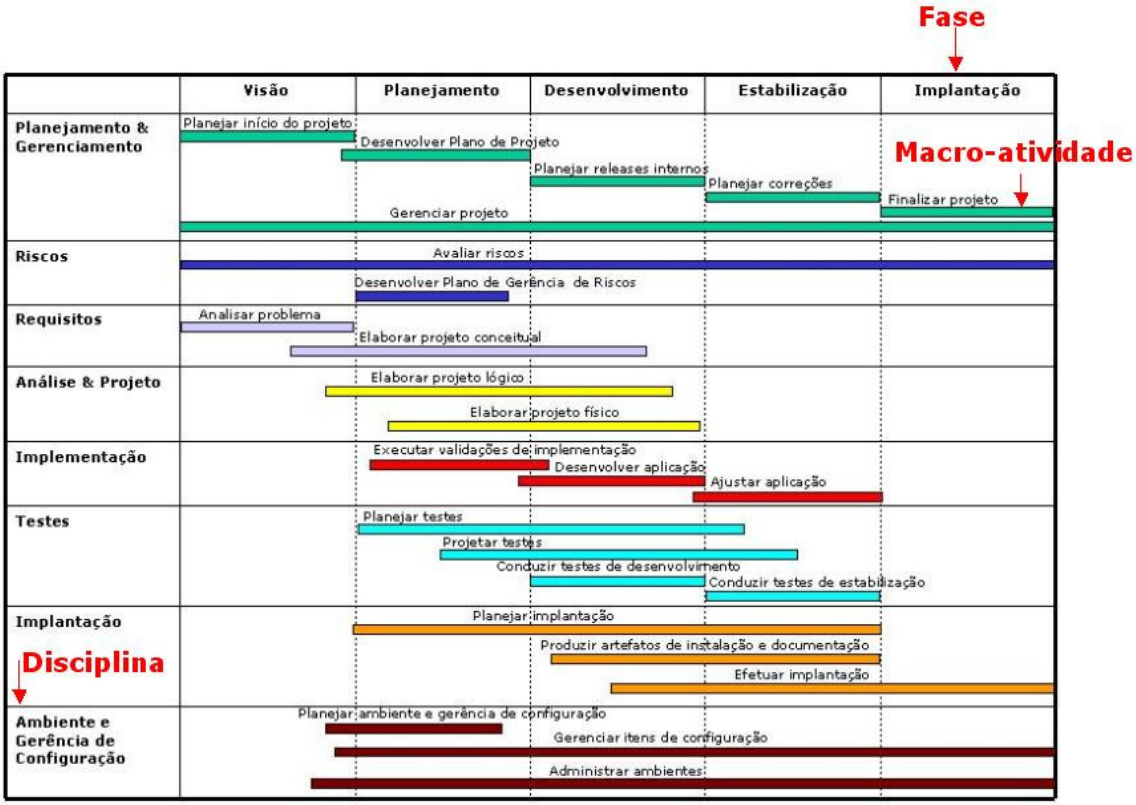

Figura 46 – Atividades de acordo com o tempo e as áreas de conhecimento na Pro.NET

Cada atividade possui um objetivo, um responsável, passos, artefatos de entrada e saída e, possivelmente, guias. O objetivo indica o que a realização daquela atividade pretende atingir dentro do contexto de desenvolvimento do projeto. O responsável é um membro do Modelo de Equipe que irá liderar a execução da atividade, respondendo pelo seu sucesso. Os passos são a divisão do trabalho a ser realizado na atividade em unidades menores. Os artefatos de entrada são necessários à execução da atividade, que gera artefatos de saída. Os guias contêm um conjunto de orientações para facilitar a execução da atividade.

A metodologia fornece também guias e *templates* para artefatos, sendo estes últimos bastante utilizados no projeto VHS-AM. Os guias apresentam boas práticas no contexto de determinadas atividades, enquanto *templates* auxiliam na geração dos artefatos durante a execução de um projeto. Tais artefatos devem ser permanentemente atualizados, refletindo as mudanças e o próprio amadurecimento do projeto.

7. CONCLUSÃO

Esta seção apresenta uma reflexão crítica sobre os resultados até então obtidos no projeto VHS-AM, identificando suas virtudes e deficiências. Adicionalmente, são apontados os caminhos futuros a serem seguidos pelo aluno deste Trabalho de Graduação no futuro, não limitados à esfera dos projetos VHS/VHS-AM.

7.1. Considerações sobre o Cronograma do Projeto VHS-AM

Um fato extremamente importante e que deve ser exposto com clareza é que o projeto VHS-AM está atrasado em relação ao seu cronograma original. Planejamentos iniciais previam a conclusão do segundo *release* interno do projeto antes do prazo de finalização para este Trabalho de Graduação, porém, isto não aconteceu.

De modo a rastrear a causa desse atraso de cronograma, o aluno deste Trabalho de Graduação elaborou uma estimativa para a distribuição das atividades do projeto VHS-AM no tempo, classificando-as em três grupos: Configuração de Ambiente, Elaboração de artefatos e Implementação/Testes. O resultado pode ser visualizado no Gráfico 1.

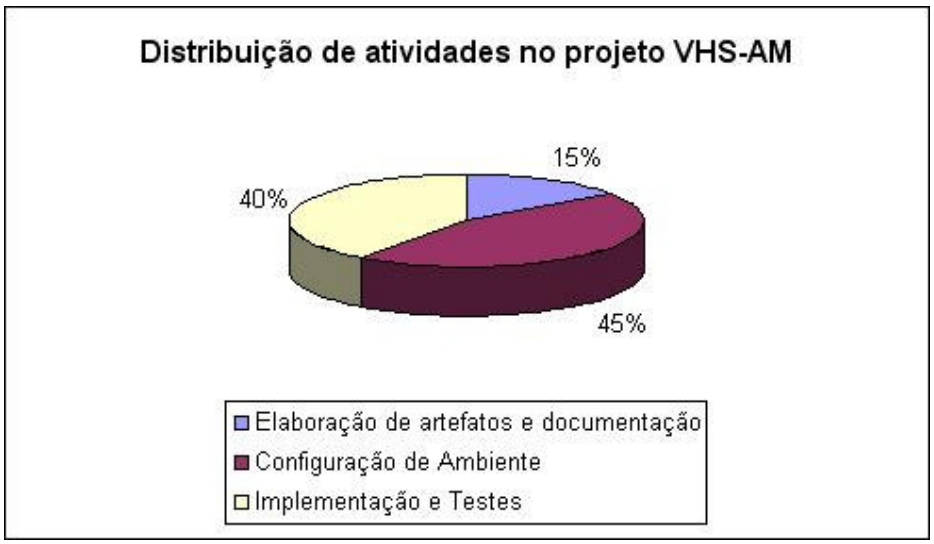

Gráfico 1 - Estimativa percentual para as atividades no projeto VHS-AM

Através do gráfico elaborado, não é difícil perceber algo incomum: o elevado tempo dedicado exclusivamente à configuração de ambiente no projeto. A princípio, pode-se imaginar que a causa de tal fato consistiu na indisponibilidade das máquinas do Centro de Informática ou no incidente da formatação do laboratório C6 sem aviso prévio à equipe do projeto. Entretanto, tais acontecimentos não justificam os dados exibidos na estimativa acima. A real dificuldade enfrentada pela equipe do projeto se refere ao número razoavelmente alto

de ferramentas e tecnologias que precisaram ser integradas antes do início da Fase de Desenvolvimento. A listagem abaixo, por exemplo, retrata um pouco das dificuldades encontradas em relação à configuração de ambiente, que retardaram o desenvolvimento do projeto:

- Compilação do GHC e de um *runtime system* específico para o mesmo (ver Capítulo 5);
- Compilação de demais ferramentas do repositório Haskell, como Happy, Alex e HDirect (este último, inclusive, apresentando algumas "quebras de build" e *bugs* em tempo de execução);
- Adaptação, para o Visual Studio .NET 2003, de um código-fonte originalmente desenvolvido para a versão 2002 do Visual Studio .NET;
- Necessidade de utilização de um ambiente Unix (mesmo que simulado em uma plataforma Windows) para compilação não apenas das ferramentas do repositório da Haskell.org, como também do código do próprio Haskell *VSPakage*.

Além disso, as etapas de implementação também foram dificultadas pelo fato do VSIP ser protegido por licença, o que impede a publicação de experiências de outros programadores pela Web. Por fim, aliada a essa escassez de experiências e exemplos na Web, a inconsistência e desatualização da documentação interna do próprio VSIP também dificultaram o trabalho da equipe.

Os membros do projeto VHS-AM, entretanto, acreditam que estas dificuldades fazem parte de uma etapa já em superação, estando os problemas cada vez mais mantidos sob controle. O processo de compilação das ferramentas do repositório da Haskell.org, por exemplo, foi dominado com mais segurança, assim como modelo de funcionamento do VSIP. Espera-se, portanto, que as atividades alocadas no futuro se dêem em um ambiente mais estável, visando apenas ao enriquecimento das funcionalidades do Haskell *VSPackage* em desenvolvimento.

7.2. Reflexão sobre os Resultados Obtidos

Embora o surgimento de algumas dificuldades tenha atrasado o cronograma original do projeto, pode-se dizer que o resultado obtido até então é satisfatório. Utilizando um conjunto de tecnologias pioneiras, nunca antes integradas, os projetos VHS/VHS-AM estão validando com sucesso a utilização de Haskell não apenas no VS.NET, IDE extremamente poderosa, como também em demais ferramentas, como o Assignment Manager.

Diferentemente de todas as demais propostas de IDE apresentadas para Haskell até então, este projeto está contribuindo para a definição de um conjunto de interfaces a ser exposto pelo GHC, que disponibilizará para o VS.NET exatamente o que a IDE deseja obter. No futuro, inclusive, isto contribuirá para que outras IDEs reutilizem esse conjunto de interfaces e possam oferecer um suporte mais adequado ao programador Haskell.

Por fim, a validação da metodologia Pro.NET para um projeto externo ao Centro de Tecnologia XML do Recife está sendo um trabalho bastante interessante. Uma base de conquistas, desafios e lições aprendidas em relação ao uso da metodologia está sendo montada, de modo a permitir que a Pro.NET seja melhorada em um futuro próximo.

7.3. Trabalhos Futuros

Dar seqüência ao projeto VHS-AM é uma atividade natural a ser efetuada pelo aluno deste Trabalho de Graduação. Depois da conclusão do segundo e terceiro *releases* internos, deve ser avaliada a possibilidade de uma nova iteração no projeto, abordando funcionalidades que ficaram de fora desta primeira iteração, como o suporte à depuração, por exemplo. Mesmo que elas não sejam implementadas como parte do escopo do projeto VHS-AM, há a possibilidade delas serem realizadas como parte do projeto VHS.

Entretanto, popularizar a linguagem Haskell e ampliar seu campo de atuação, o verdadeiro sentido deste trabalho, não está limitado pela integração a uma IDE ou qualquer outra ferramenta. Existem várias restrições para a linguagem, assim como identificado no Capítulo 2, que podem e devem ser atacadas no futuro. Desse modo, existem muitas possibilidades diferentes de contribuição para o crescimento de Haskell como linguagem e como comunidade.

Uma extensão deste trabalho, que visa tal objetivo, consiste em permitir a geração de código gerenciado, para a plataforma .NET, a partir de código Haskell. Este novo "Visual Haskell .NET" inserirá Haskell como uma opção para a atual estratégia de desenvolvimento da Microsoft, que consiste na criação de um cenário em que todas as linguagens possam conversar entre si.

Costuma-se dizer que esse cenário apresenta uma característica interessante: "a escolha da linguagem de programação não mais determina o que é possível de ser implementado" [11]. Entretanto, apesar de qualquer produto final, nesse cenário, poder ser implementado por qualquer linguagem, restará uma questão: que linguagem o fará com mais produtividade? Espera-se que, de uma maneira ou de outra, "Visual Haskell .NET" contribua para a resposta, assim como este trabalho vem contribuindo para que Haskell seja encarada como uma alternativa concreta para a implementação ágil de projetos de desenvolvimento de software.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] 4Reference.Net, *The BASIC Programming Language*, disponível em http://www.4reference.net/encyclopedias/wikipedia/True_BASIC_programming_language.html (25/03/2004)

[2] Active State.com, *Visual Perl*, disponível em http://www.activestate.com/Products/Visual_Perl/?_x=1 (25/03/2004)

[3] Active State.com, *Visual Python*, disponível em http://www.activestate.com/Products/Visual_Python/?_x=1 (25/03/2004)

[4] AllenD's weblog, *Can Add-ins & managed packages live together?*, disponível em http://blogs.msdn.com/allend/archive/2004/03/31/104829.aspx (26/03/2003)

[5] Bagley, D. *Computer Language Shootout Scorecard*, disponível em http://www.bagley.org/~doug/shootout/craps.shtml (26/03/2004)

[6] Barry & Associates, Inc, *Web Services and Service-Oriented Architectures*, disponível em http://www.service-architecture.com/ (24/03/2004)

[7] Boekhoudt, G. *The Big Theory of IDEs*, ACM Queue, p. 74-83, outubro/2003.

[8] BYTE.COM. *A Brief History of Programming Languages*, disponível em http://www.byte.com/art/9509/sec7/art19.htm (29/03/2004)

[9] C# Corner*, Cobol.NET Programming*, disponível em http://www.c-sharpcorner.com/cobolnet/code.asp (23/03/2004)

[10] C2.com (Cunningham & Cunningham, Inc). *Haskell Language*, disponível em http://c2.com/cgi/wiki?HaskellLanguage (23/03/2004)

[11] C2.com (Cunningham & Cunningham, Inc), *Is Visual Basic Preferred Over Csharp At Dot Net*, disponível em http://c2.com/cgi/wiki?IsVisualBasicPreferredOverCsharpAtDotNet (22/03/2004)

[12] C2.com (Cunningham & Cunningham, Inc), *Literate Programming*, disponível em http://c2.com/cgi/wiki?LiterateProgramming (23/03/2004)

[13] Caml.org, *The Caml language*, disponível em http://caml.inria.fr/ (23/03/2004)

[14] Cunha C., *Pro.NET- Processos de Desenvolvimento de Software para a Plataforma .NET*, Encontro da Qualidade e Produtividade em Software (EQPS), Fortaleza, setembro/2003.

[15] Donzeau-Gouge V., Huet G., Kahn G., Lang, B. *Programming environments based on structured editors: The MENTOR experience*. Interactive Programming Environments, INRIA, Paris, 1980.

[16] *Eclipse.org*, disponível em http://www.eclipse.org (25/03/2004)

[17] Engels g., Lewerentz c., Nagl m., Schäfer w., Schürr a., *Building integrated software development environments. Part I: tool specification*, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 1, no. 2 , p. 135-167, abril/1992

[18] Erlang.org, *Erlang*, disponível em http://www.erlang.org/ (23/03/2004)

[19] Faqs.org, *What is black box/white box testing?*, disponível em http://www.faqs.org/faqs/software-eng/testing-faq/section-13.html (27/03/2004)

[20] Finne S., Leijen D, Meijer E, Peyton-Jones S., *Calling Hell From Heaven And Heaven From Hell*. International Conference on Functional Programming, 1999.

[21] Finne, S. *Haskell Direct*, disponível em http://www.haskell.org/hdirect (24/03/2004).

[22] Frenzel L., *Haskell support for the Eclipse IDE*, disponível em http://leiffrenzel.de/eclipse/eclipsefp.html (25/03/2004)

[23] Furtado, A. W. B., Santos, A. L. M. *FunGEn - A Game Engine for Haskell*, 1o Workshop de Jogos e Entretenimento Digital (WJogos), outubro/2002

[24] Gehringer M., London J., *Odisséia Digital*, Encarte especial da Revista Superinteres-sante, Editora Abril, abril/2001,

[25] GNU.org, *GCC Homepage*, disponível em http://gcc.gnu.org (28/03/2004)

[26] Griffiths I., Flanders J., Sel C., *Mastering Visual Studio .NET*, 1a edição, editora O'Reilly, março/2003.

[27] Haaften R*., JCreator with Haskell Support*, disponível em http://www.students.cs.uu.nl/people/rjchaaft/JCreator/ (25/03/2004)

[28] Hampton, W. *What is a Tarball? How do I extract it?,* disponível em http://www.redhat-linux.com.my/faq/common15.html (27/03/2004)

[29] Haskell.org, *A Gentle Introduction to Haskell, Version 98*, disponível em http://www.haskell.org/tutorial/ (23/03/2004)

[30] Haskell.org, *Alex: A lexical analyser generator for Haskell*, disponível em http://www.haskell.org/alex (28/03/2004)

[31] Haskell.org, *Building the Glasgow Functional Programming Tools Suite: Notes for building under Windows*, disponível em http://www.haskell.org/ghc/docs/latest/html/building/winbuild.html (28/03/2004)

[32] Haskell.org, *Building the Glasgow Functional Programming Tools Suite: The Makefile Architectur*e, disponível em http://www.haskell.org/ghc/docs/latest/html/building/sec-makefile-arch.html (28/03/2004)

[33] Haskell.org*, Happy: The Parser Generator for Haskell*, disponível em http://www.haskell.org/happy (28/03/2004)

[34] Haskell.org, *Haskell: A Purely Functional Language*, disponível em http://www.haskell.org (23/03/2004)

[35] Haskell.org, *Hugs Online*, disponível em http://www.haskell.org/hugs/ (25/03/2004)

[36] Haskell.org, *Libraries and Tools For Haskell: Integrated Development Environments*, disponível em http://www.haskell.org/libraries/#ide (22/03/2004)

[37] Haskell.org, *The Glorious Glasgow Haskell Compiler*, disponível em http://www.haskell.org/ghc/ (22/03/2004)

[38] Haskell.org. *About Haskell*, disponível em http://www.haskell.org/aboutHaskell.html (23/03/2004).

[39] IBM.com, *Rational Unified Process*, disponível em http://www-306.ibm.com/software/awdtools/rup (25/03/2004)

[40] IDM Computer Solutions, Inc., *UltraEdit-32*, disponível em http://www.ultraedit.com/products/index.html (25/03/2004)

[41] Ierusalimschy R., Cerqueira R., *Lua.NET: Integrating Lua with Rotor*, disponível em http://www.tecgraf.puc-rio.br/~rcerq/luadotnet/ (22/03/2004)

[42] Instituto Superior Técnico de Portugal, *O Movimento de Software Livre e o Sistema Operativo GNU/Linux*, disponível em http://www.tagus.ist.utl.pt/Stallman/indexportugues.htm (24/03/2004)

[43] KDE.org, *KDE Homepage*, http://www.kde.org (25/03/2004)

[44] KDevelop.org, *KDevelop Home*, disponível em http://www.kdevelop.org (25/03/2004)

[45] Kinnersley, B. *The Language List*, disponível em http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm (29/03/2004)

[46] Lévénez, E. *Computer Languages History*, disponível em http://www.levenez.com/lang/ (30/01/2004)

[47] Lisp.org, *Association of Lisp Users*, disponível em http://www.lisp.org/alu/home (23/03/2004)

[48] MicroLink.com., *A Superioridade de um ou a Desmitificação do Outro?*, disponível em http://www.microlink.com.br/~buick/dragons/freevslinux.html (24/03/2004)

[49] Microsoft MSND, *Visual Studio Industry Partner Distribution Agreement*, disponível em http://www.vsipdev.com/LicenseAgreement.aspx (27/03/2004)

[50] Microsoft PressPass, *Microsoft Selects Five Universities To Enhance Visual Studio .NET 2003 Academic*, disponível em http://www.microsoft.com/presspass/press/2003/nov03/11-04UniversitiesVSTOPR.asp (21/03/2004)

[51] Microsoft Research, *Microsoft Research Cambridge*, disponível em http://research.microsoft.com/cambridge/ (22/03/2004)

[52] Microsoft Research, *Simon Marlow @ Microsoft Cambridge Research*, disponível em http://research.microsoft.com/~simonmar/ (22/03/2004)

[53] Microsoft Visual Studio Developer Center, *Automation and Extensibility Overview*, disponível em http://msdn.microsoft.com/vstudio/extend/?pull=/library/en-us/dv_vstechart/html/vsoriautomationextensibilityoverview.asp (26/03/2003)

[54] Microsoft, *ASP.NET Starter Kits*, disponível em http://www.asp.net/Default.aspx?tabindex=9&tabid=47 (21/03/2004)

[55] Microsoft, *Internet Information Services*, disponível em http://www.microsoft.com/WindowsServer2003/iis/default.mspx (25/03/2004)

[56] Microsoft, *Microsoft .NET Framework Developing Center: Technology Overview*, dis-ponível em http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx (25/03/2004)

[57] Microsoft*, Microsoft Message Queuing (MSMQ) Center*, disponível em http://www.microsoft.com/windows2000/technologies/communications/msmq/default.asp (25/03/2004)

[58] Microsoft, *Microsoft Shared Source Initiative*, disponível em http://www.microsoft.com/resources/sharedsource/default.mspx (21/03/2004)

[59] Microsoft, *Microsoft SQL Server 2000 Desktop Engine*, http://www.microsoft.com/sql/msde/default.asp (25/03/2004)

[60] Microsoft, *Microsoft Visual Studio Developer Center*, disponível em http://msdn.microsoft.com/vstudio (25/03/2004)

[61] Microsoft, *MSDNAA ASSC Project Space Listings*, disponível em http://www.msdnaa.net/assc/projects/ (25/03/2004)

[62] Microsoft, *Product Overview for Visual Studio .NET 2003*, disponível em http://msdn.microsoft.com/vstudio/productinfo/overview/default.aspx (25/03/2004)

[63] Microsoft, *Recife inaugura Centro de Tecnologia XML*, disponível em www.microsoft.com/brasil/pr/2003/xml_recife.asp (28/03/2004)

[64] Microsoft, *The Official Microsoft ASP.NET Site*, disponível em http://www.asp.net (21/03/2004)

[65] Microsoft, *Visual Studio .NET Academic Tools Source Licensing Pro*-gram , disponível em http://www.microsoft.com/resources/sharedsource/Licensing/VSAcademic.mspx (21/03/2004)

[66] MinGW.org, *Minimalist GNU For Windows*, disponível em http://www.mingw.org (02/03/2004)

[67] *Monash University*, disponível em http://www.monash.edu.au (25/03/2004)

[68] MSDN Brasil, *ASP.NET Starter Kits*, acessível em http://www.msdnbrasil.com.br/tecnologias/aspnet/starterkits/ (21/03/2004)

[69] Network Integrated Access (Netia.com), *The Next Major Platform Shift: Java Computing Changes Everything*, disponível em http://www.netia.com/Sun_Java_Sea_Change.pdf (23/03/2004)

[70] Peyton-Jones S., Meijer E., Leijen D. *Scripting COM components in Haskell*. IEEE Fifth International Conference on Software Reuse, Vancouver, BC, 1998

[71] Sebesta R. W., *Concepts of Programming Languages*, The Benjamin/Cummings Publishing Company, Inc, 1993

[72] SMLNJ , *Standard ML'97*, disponível em http://www.smlnj.org/sml97.html (23/03/2004)

[73] Software Technology Research Group, *Homepage of Clean*, disponível em http://www.cs.kun.nl/~clean/ (23/03/2004)

[74] Steele K. *Eclipse: The IDE for Everything and Nothing in Particular*, The Cursor: Official Publication of the Software Association of Oregon, dezembro/2003.

[75] Teitelbaum, T., Reps, T. *The Cornell program synthesizer*: A syntax-directed programming environment. Communications of the ACM, vol.24 no. 9, p. 563-573, setembro/1981

[76] The Lambda Complex, *Why does Haskell matter?*, disponível em http://www.haskell.org/complex/why_does_haskell_matter.html (23/03/2003)

[77] *The University of Hull*, disponível em www.hull.ac.uk (25/03/2004)

[78] *Univeristy of Cambridge*, SML.NET, disponível em http://www.cl.cam.ac.uk/Research/TSG/SMLNET (25/03/2004)

[79] *Universidade Estadual Paulista Julio de Mesquita Filho*, disponível em http://www.unesp.br/ (25/03/2004)

[80] Wadler, P. *Why no one uses functional languages*, ACM SIGPLAN Notices, agosto/1998

[81] Webopedia.com, *What is API?,* disponível em http://www.webopedia.com/TERM/A/API.html (23/03/2004)

[82] *Yale University*, disponível em http://www.yale.edu (25/03/2004)

[83] DinosaurCompilertools.net, *The Lex & Yacc Page*, disponível em http://dinosaur.compilertools.net (24/03/2004)

[84] CVSHome.org, *Concurrent Versions System*, disponível em http://www.cvshome.org (24/03/2004)

[85] Object Management Group, *The Common Object Request Broker: Architecture and Specification*. pg. 66, julho, 2002.

[86] Microsoft.com, *Microsoft Solutions Framework*, disponível em http://www.microsoft.com/msf (24/03/2004)

Haskell: that's where I just curry until fail, unwords any error, drop all undefined, maybe break, otherwise in sequence span isControl and take max $, id: (d:[])