Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

VSFocus: Orientação a Tarefas para Microsoft Visual Studio

Trabalho de Graduação em Engenharia de Software

Aluno: João Augusto B. C. Alves (jabca@cin.ufpe.br)

Orientador: André Luis Medeiros dos Santos (alms@cin.ufpe.br)

Co-orientador: Ryan Leite Albuquerque (ryan@cesar.org.br)

Recife, 22 de Agosto de 2007

“Você vê coisas e você diz, ‘Por que?’. Mas eu sonho
coisas que nunca existiram e digo, ‘Por que não?’

George Bernard Shaw

Agradecimentos

Em primeiro lugar à minha mãe e meus tios, por todo o sacrifício, amor, carinho, orações, cartuchos, e pelos 185 quilômetros.

Aos amigos fantásticos: a Thaíza, a Carol e o Zé, por toda a motivação (lê-se pentelhação) fornecida durante todo o decorrer desta jornada.

À menina Mona pelos momentos de inspiração e pelos ISOs, ABNTs, NBRs e similares.

Aos meus orientadores pela extrema confiança, paciência e compreensão.

A Renan, o neural. E aos compilados tio Ewma e tia Nicky.

Enfim, a todos vocês que participaram ativamente, passivamente ou de outras maneiras alternativas na realização deste trabalho. Sem vocês nada teria valido a pena.

Resumo

Com as recentes variações de preço, as novas edições do Visual Studio Team System, um dos ambientes de desenvolvimento mais utilizados no mercado mundial de software, estão se tornando cada vez mais inacessíveis para pequenas empresas, além de fornecer recursos muitas vezes desnecessários para a realidade dessas organizações. Neste contexto, o projeto propõe o desenvolvimento de um protótipo de ferramenta para integração entre o ambiente Visual Studio 2005 e sistemas de *issue tracking*, tomando como inspiração o projeto Mylyn. Possibilitando aos desenvolvedores um melhor gerenciamento de suas tarefas e mais agilidade na comunicação, além de ajudar a manter o foco no contexto das atividades em andamento. Facilitando assim a adoção de processos de gerência de configuração em um ambiente ágil de desenvolvimento.

Palavras-chave: Visual Studio 2005, Eclipse, Mylyn, Trac, integração, C#, .NET

Sumário

Índice de Figuras

Índice de Tabelas

1 Introdução

Enquanto sistemas de *software*, *frameworks* e ferramentas continuam a se expandir indefinidamente, a habilidade de um desenvolvedor para processar informação tem um limite prático. A maioria dos ambientes de desenvolvimento (*IDEs*) modernos tratam esta questão com recursos para compilação incremental, visualizadores de estrutura, e busca textual. Apesar de estes mecanismos facilitarem a navegação em sistemas complexos, eles não dão nenhuma ajuda aos desenvolvedores para gerenciar o enorme volume de informação que precisa ser processado para concluir uma atividade de programação (1). Como resultado, as interfaces dessas *IDEs* tornam-se cada vez mais sobrecarregadas, dificultando o trabalho e diminuindo a produtividade.

Além disso, vale mencionar o importante papel da gerência de configuração no contexto das metodologias ágeis de desenvolvimento. Freqüentemente, gerência de configuração e controle de versão são consideradas pelos desenvolvedores atividades complexas em um processo de desenvolvimento, que podem atrapalhar o "trabalho de verdade" da programação. Na realidade, é um bom processo de gerência de configuração, aliado a um conjunto de ferramentas adequado, o que permite um trabalho em equipe efetivo (2).

Neste âmbito, é crucial notar a importância da integração entre ferramentas para o sucesso de uma equipe de desenvolvimento ágil[1]. A integração mantém o ambiente de software mais padronizado e pode reduzir substancialmente o nível de mudanças de contexto, promovendo assim um maior desempenho em geral (3). Pondo em termos práticos: abrir um navegador web apenas para marcar um *bug* como corrigido pode se tornar algo desestimulante, além de ser um convite à perda de tempo (4).

Com as recentes variações de preço, as novas edições do Visual Studio Team System[2] (VSTS), um dos ambientes de desenvolvimento mais utilizados no mercado mundial de software (5), estão se tornando cada vez mais inacessíveis para pequenas empresas, além de fornecerem recursos muitas vezes desnecessários para a realidade dessas organizações. Porém, versões mais restritas, como o Visual Studio Standard Edition[3] (VSSE), não apresentam recursos para trabalho em equipe, nem mesmo serviços mais básicos como controle de versão, rastreamento de tarefas (*issue tracking*) e testes unitários.

Por outro lado, muitos desses serviços podem ser providos através de componentes de software livre/gratuito (6), que em sua maior parte não apresentam integração com *IDEs* comerciais. Exemplos populares desses sistemas são: o Subversion[4], que possui certo nível de

[1] http://en.wikipedia.org/wiki/Agile_software_development
[2] http://msdn.microsoft.com/vstudio/teamsystem/
[3] http://msdn2.microsoft.com/en-us/vstudio/aa718671.aspx
[4] http://subversion.tigris.org/

integração com o VSSE; o Trac[5]; e o NUnit[6], que também possui *plug-ins* para integração com o VSSE.

1.1 Objetivos

Neste contexto, o projeto propõe o desenvolvimento de um protótipo de ferramenta para integração entre o ambiente Visual Studio 2005[7] (VSSE05) e sistemas de *issue tracking*, tomando como inspiração o projeto Mylyn[8]. Possibilitando aos desenvolvedores um melhor gerenciamento de suas atividades e mais agilidade na comunicação, além de ajudar a manter o foco no contexto das tarefas em andamento (4) (7).

O protótipo proposto deve incluir as seguintes características:

- Gestão de tarefas
  - Gestão de repositórios
  - Painel de tarefas
  - Busca e filtragem (desejável)
- Suporte a *issue trackers*
  - Trac
    - Consultas
    - Edição integrada (desejável)
    - Anexos (desejável)
    - Modo *offline* e Notificações (desejável)
  - Bugzilla[9] (desejável)
  - Google Code Hosting[10] (desejável)
  - Mantis[11] (desejável)
  - JIRA[12] (desejável)

1.2 Metodologia Aplicada

Para a realização deste trabalho foram necessários, além de um levantamento bibliográfico e das etapas tradicionais da produção de software, estudos específicos sobre técnicas e ferramentas utilizadas no desenvolvimento do protótipo. Tais atividades são sumarizadas a seguir:

---

[5] http://trac.edgewall.org/
[6] http://www.nunit.com/index.php
[7] http://msdn2.microsoft.com/en-us/vstudio/aa973782.aspx
[8] http://www.eclipse.org/mylyn/
[9] http://www.bugzilla.org/
[10] http://code.google.com/hosting/
[11] http://code.google.com/hosting/
[12] http://www.atlassian.com/software/jira/

- Pesquisa
    - Revisão bibliográfica
    - Análise em detalhes da ferramenta Mylyn
    - Análise da plataforma Visual Studio e seus recursos de extensibilidade[13]
    - Revisão da linguagem C#[14] e do .NET Framework[15]
- Desenvolvimento
    - Definição e priorização dos requisitos
    - Design dos módulos principais
    - Implementação
- Documentação e redação da monografia

1.3 Organização do Trabalho

Os capítulos 2, 3 e 4 apresentam o contexto tecnológico relacionado ao projeto VSFocus. No capítulo 2 é apresentada uma visão geral da ferramenta Mylyn, com seus princípios e características centrais. O capítulo 3 mostra a IDE Visual Studio 2005 e suas possibilidades de extensão. O capítulo 4 apresenta o funcionamento básico do *issue tracker* Trac, escolhido como alvo para integração neste trabalho. Fechado o contexto tecnológico necessário para a realização do projeto, o capítulo 5 documenta as estratégias utilizadas e os resultados obtidos no desenvolvimento da ferramenta VSFocus. Finalmente, o capítulo 6 apresenta algumas reflexões sobre os resultados obtidos e sugestões de trabalhos futuros.

[13] http://msdn.microsoft.com/vstudio/extend/
[14] http://msdn.microsoft.com/vcsharp/
[15] http://msdn.microsoft.com/netframework/

2 Mylyn

O Mylyn é uma interface de orientação a tarefas para Eclipse[16] que reduz a sobrecarga de informação e facilita o trabalho com múltiplas tarefas. Ele faz isso tornando as tarefas uma parte de primeira classe do Eclipse, e integrando recursos de edição rica e *offline* para repositórios como Bugzilla[17], Trac, e JIRA[18]. Uma vez que as tarefas estão integradas, o Mylyn monitora a atividade de trabalho do usuário para identificar informações relevantes à tarefa executada no momento, e usa esse contexto da tarefa para focar a interface do Eclipse nas informações interessantes, esconder as desinteressantes, e automaticamente encontrar o que está relacionado. Isso põe as informações necessárias para concluir o trabalho ao alcance das mãos e aumenta a produtividade reduzindo buscas, rolagem e navegação. Tornando o contexto da tarefa explícito, o Mylyn também facilita a gestão de múltiplas tarefas, o planejamento, o reuso de esforços passados e o compartilhamento de experiências (8).

O funcionamento do Mylyn baseia-se na interação do usuário, monitorando o Eclipse e capturando-a em um contexto de tarefa. Artefatos do sistema como arquivos, tipos (classes), métodos e atributos, recebem a atribuição de um grau de interesse baseado no quão recentemente e freqüentemente o usuário interagiu com eles (9).

2.1 Motivação

Quanto maior é o sistema de informação a ser acessado pelo profissional do conhecimento, mais onerosas se tornam a perda de contexto e a sobrecarga de informação. No domínio de conhecimento do desenvolvimento de software, os sistemas de informação de interesse são as aplicações e *frameworks* que desenvolvedores criam e integram. Muitos *frameworks* e aplicações modernas consistem de milhões de linhas de código. Ambientes de desenvolvimento atuais, como o Eclipse e o Visual Studio, deixam estes milhões de linhas de código instantaneamente acessíveis ao programador através de indexações sofisticadas e ferramentas de busca. Porém, quando está trabalhando em uma tarefa específica, como adicionar um recurso a uma aplicação, o programador está interessado apenas em uma porção bastante pequena do código sobre a qual está desenvolvendo. Ele pode tentar usar ferramentas de consulta para ajudar a identificar a informação relevante à tarefa, mas como essa informação geralmente atravessa toda a estrutura do sistema, é difícil formular consultas adequadas em um intervalo de tempo razoável. Outra alternativa seria etiquetar partes da estrutura com marcadores ou outros mecanismos de anotação, mas etiquetagens e buscas constantes são fatigantes. Longas listas de etiquetas podem também contribuir para a

[16] http://www.eclipse.org/
[17] http://www.bugzilla.org/
[18] http://www.atlassian.com/software/jira/

sobrecarga de informação quando uma nova tarefa de maior prioridade requer atenção, visto que elas podem não ser relevantes para a nova tarefa (10).

2.2 Um exemplo

Nesta seção, para dar uma visão geral e demonstrar algumas funcionalidades da ferramenta, apresentamos dois cenários de uso do Eclipse (10). O primeiro cenário exemplifica o modelo tradicional do ambiente aplicado a um projeto razoavelmente complexo, enquanto o segundo mostra o Eclipse em ação dispondo dos recursos providos pelo Mylyn.

2.2.1 Antes do Mylyn

Considere o exemplo concreto de um programador tentando descobrir porque alguns dos *test cases* para desserialização estão falhando no Web Services Invocation Framework (WSIF)[19]. Para completar essa tarefa, o programador precisa examinar os *test cases*, as classes que estão apresentando falhas ao desserializar, e a política de desserialização empregada no sistema. Usando o Eclipse, o programador decide encontrar todos os subtipos na base de código do WSIF que implementam a interface `Serializable`, e inspecionar os métodos de acesso nessas classes. A Figura 1: Sobrecarga de informação no Eclipse mostra uma captura de tela do Eclipse no momento em que o programador está na metade do caminho para concluir a tarefa.

O painel *Package Explorer* (navegador de pacotes) se tornou difícil de usar, pois inclui milhares de nós — resultado de somente alguns poucos cliques de navegação pelos arquivos do projeto e classes das bibliotecas relacionadas. Relacionamentos hierárquicos não podem mais ser visualizados sem uma rolagem manual pela árvore.

1. Em parte, graças à facilidade que o Eclipse provê para navegar relacionamentos estruturais, o número de editores abertos pode rapidamente se exceder às dezenas quando se trabalha em uma tarefa moderadamente complexa, tornando a lista de editores uma representação pobre dos arquivos atualmente relevantes à tarefa.
2. A busca Java[20] do Eclipse procurando por referências a `Serializable` no projeto retornou 144 itens. Não existe uma maneira conveniente para procurar apenas pelos elementos relacionados à tarefa de corrigir os *test cases* que falharam. Ao invés disso, a lista do resultado da busca requer uma inspeção manual para encontrar os elementos de interesse.
3. Apesar do painel *Outline* (resumo) mostrar apenas a estrutura do arquivo atual, ele está sobrecarregado com dezenas de elementos que não são relevantes à tarefa.
4. O painel *Type Hierarchy* (hierarquia de tipos) exibe todos os tipos no projeto que herdam de `Serializable`, e contém milhares de elementos que precisam ser manualmente inspecionados para identificar aqueles relevantes à tarefa.

---

[19] http://ws.apache.org/wsif (1.897 classes)
[20] http://java.sun.com/javase/

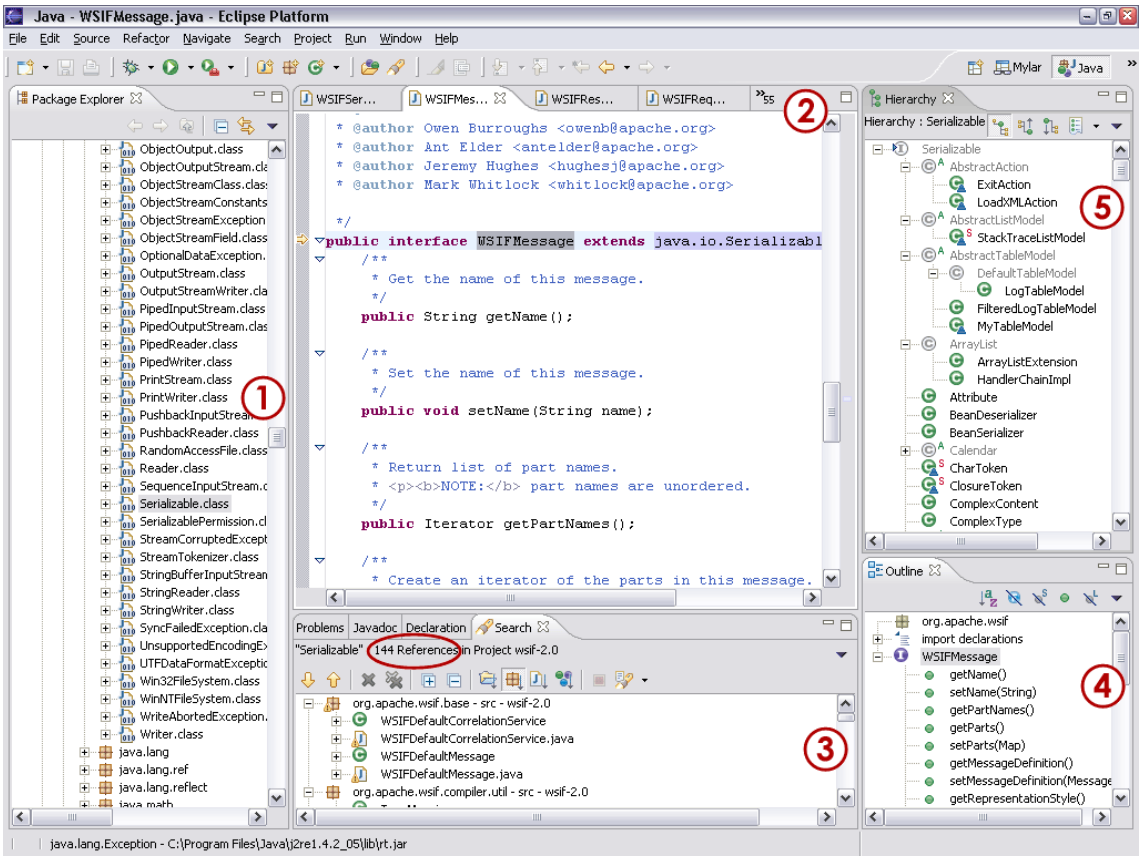

Figura 1: Sobrecarga de informação no Eclipse

Apesar desses painéis e editores estarem sobrecarregados, ainda assim eles fornecem informações úteis ao programador sobre elementos relevantes à tarefa. No entanto, assim que o programador começa a trabalhar em uma tarefa diferente envolvendo outras partes da aplicação, o contexto formado nesses painéis é substituído com os resultados de novas buscas e novas navegações.

2.2.2 Depois do Mylyn

Agora, demonstraremos o uso do Mylyn tomando como exemplo a sua própria base de código que contém cerca de mil classes Java. Considere o cenário de um programador trabalhando na seguinte tarefa: reprojetar uma política definida por uma classe, que afeta os usos desta classe no sistema. O programador utiliza o painel *Task List* (lista de tarefas) para indicar em qual tarefa ele está trabalhando no momento (através do botão de rádio exibido na Figura 2-1 à esquerda da tarefa em questão):

```
T1: Reprojetar ResourceStructureBridge
```

A tarefa envolve a identificação, inspeção e alteração de todos os clientes da classe `ResourceStructureBridge`. A ativação da tarefa faz com que o modelo de contexto de tarefa rastreie parte dos artefatos do sistema – elementos do programa e seus relacionamentos – que são acessados enquanto o programador trabalha na tarefa.

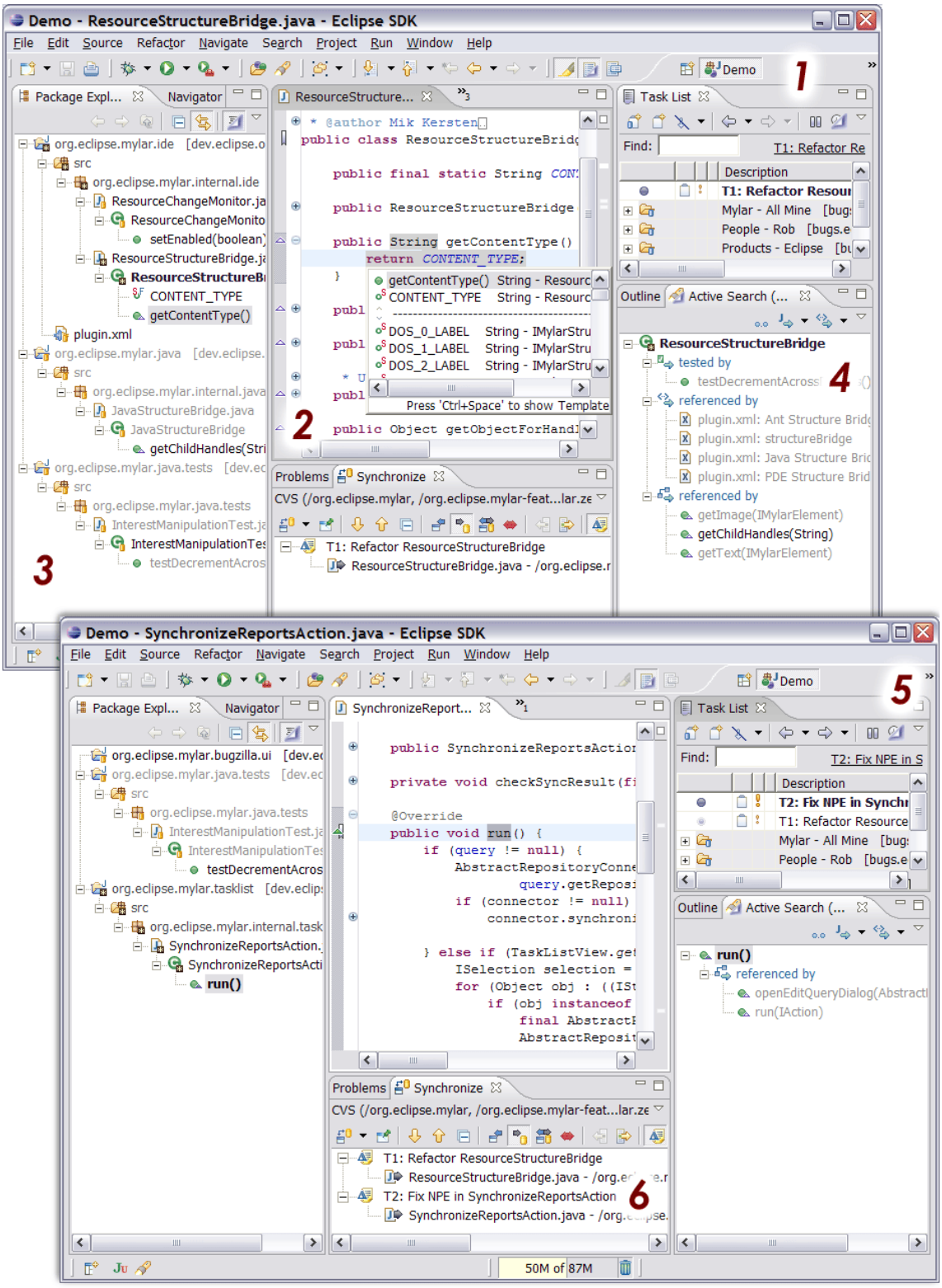

Figura 2: Contexto da tarefa ao trabalhar em T1 (acima), e logo após ativar T2 (abaixo)

O Mylyn exibe o contexto da tarefa dentro das *structure views* existentes no Eclipse com o objetivo de manter o foco do programador na tarefa em questão. O editor de código contrai todos os elementos não interessantes e ordena os itens do assistente de conteúdo, visíveis no menu *popup*, para indicar aqueles que estão no contexto da tarefa (Figura 2-2). Apesar da base

de código conter mais de mil classes e numerosos artefatos de outros tipos, apenas os artefatos determinados como relevantes ao contexto da tarefa atual estão visíveis no painel *Package Explorer* do sistema (Figura 2-3). Elementos relacionados estruturalmente que ainda não sofreram interação, mas que têm interesse previsto, aparecem no painel *Active Search* (busca dinâmica) (Figura 2-4). Quando se está trabalhando com o contexto de tarefas e a interface focada em tarefas, o ambiente inteiro se torna focado na tarefa em vez de mostrar todas as partes da estrutura do sistema, como foi visto na Figura 1. Depois que o contexto da tarefa é ativamente organizado para exibir apenas os elementos mais relevantes, o programador raramente verá uma barra de rolagem em muitos dos painéis focados na tarefa, reduzindo a sobrecarga de informação nesses painéis. Enquanto está trabalhando de forma focada em T1, um novo *bug* de alta prioridade é designado ao programador e deve ser atendido imediatamente.

`T2: Corrigir NPE[21] em SynchronizeReportsAction.`

Usando o painel *Task List* do Mylyn, o programador ativa a segunda tarefa (Figura 2-5), fazendo com que o contexto da primeira seja armazenado e todos os arquivos naquele contexto sejam fechados. À medida que o programador trabalha na nova tarefa, um novo contexto começa a se formar para a nova tarefa. O Mylyn rastreia todos os arquivos que o programador modifica ao trabalhar em cada tarefa, e agrupa as modificações de saída – a serem submetidas ao repositório de código fonte – por tarefa (Figura 2-6). Tudo o que o programador precisa fazer é retornar à primeira tarefa e reativá-la, fazendo com que os painéis e editores retornem ao estado exibido no topo da Figura 2.

2.3 Visão geral da arquitetura

Ainda segundo Kersten (10), para testar adequadamente o conceito de contexto de tarefa, foi necessária uma implementação escalável capaz de lidar com sistemas de grande porte com vários tipos de artefatos estruturados e semi-estruturados, que pudesse suportar a gestão de tarefas e integrar com ferramentas existentes usadas por profissionais do conhecimento. A arquitetura do Mylyn implementa esses critérios provendo três *frameworks* (Tabela 1: *Frameworks* do Mylyn): *Monitor*, *Context* e *Tasks*. Cada *framework* é dividido em duas partes. A parte *"core"* fornece um modelo e operações desacoplados de qualquer plataforma em particular e adequados para uso em aplicações servidor ou para embutir em outros *frameworks*, enquanto a parte *"UI"* é acoplada ao *framework* Eclipse UI.

O *framework Monitor* transforma as interações do usuário com a aplicação em eventos que são processados pelo *framework Context*. O *framework Context* implementa o modelo de contexto de tarefa, incluindo históricos de interação e operações sobre o contexto da tarefa. A parte *UI* do *framework Context* implementa a interface orientada a tarefas. A parte *core* do

[21] NPE, acrônimo para Null Pointer Exception, um popular defeito de software

framework Tasks oferece a definição de tarefas, que mapeia itens de trabalho definidos pelo usuário em tarefas. A parte *UI* do *framework Tasks* implementa os recursos de gestão de tarefas (seção 2.4).

Tabela 1: *Frameworks* do Mylyn

Framework	Core	UI	Clientes
Monitor	Histórico da interação	Monitoramento da interação	Monitores
Context	Contexto de tarefa	Interface orientada a tarefas	Bridges
Tasks	Gestão de tarefas	Painéis e editores de tarefas	Conectores

Cada framework suporta um tipo diferente de cliente. O *framework Monitor* suporta monitores que observam as interações do usuário. O *framework Context* suporta *bridges*, que fazem um mapeamento entre os elementos abstratos no modelo de contexto de tarefa e os elementos concretos em algum domínio, por exemplo, a linguagem de programação Java. O *framework Tasks* suporta conectores, que definem a unidade em que consiste uma tarefa. Por exemplo, o conector para Bugzilla define tarefas como notificações de *bugs* ou requisições de melhorias.

O diagrama da Figura 3 demonstra o modelo de componentes do Mylyn, com cada elemento correspondendo a um plug-in OSGi[22] do Eclipse.

[22] http://www.eclipse.org/equinox/

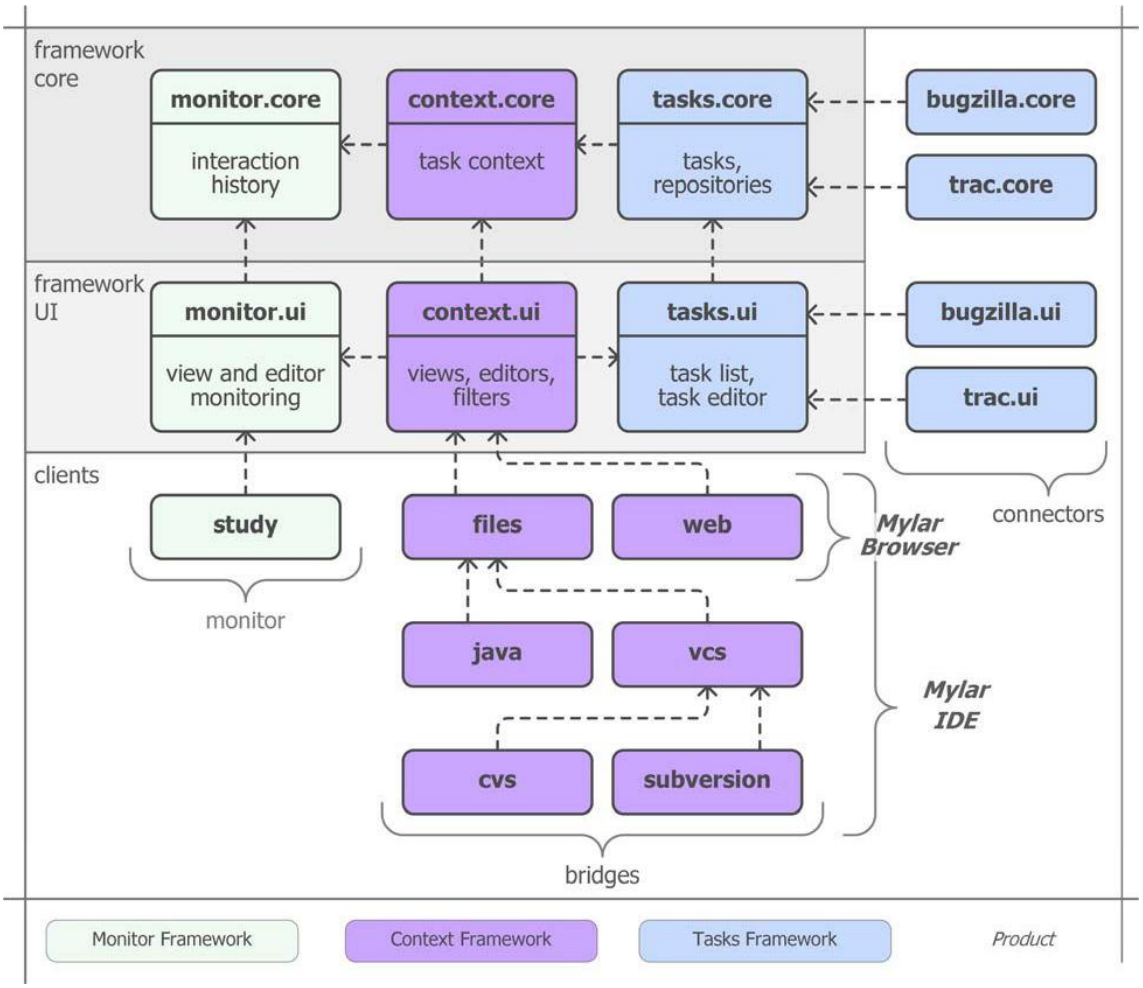

Figura 3: Arquitetura do Mylyn

2.4 Gestão de tarefas

O Mylyn fornece suporte para se trabalhar com dois tipos de tarefas (1):

- **Tarefas locais** são armazenadas no próprio *workspace* do Eclipse e apresentam os recursos básicos de gestão, como o agendamento. Tarefas locais são privadas e visíveis apenas para o programador.
- **Tarefas de repositório** são armazenadas em um repositório de tarefas que corresponde a uma aplicação servidor externa como um *issue tracker*. Tarefas de repositório são freqüentemente compartilhadas entre colaboradores, ainda que mantendo as mesmas facilidades das locais como agendamento e anotações.

2.4.1 Tarefas locais

Tarefas locais podem ser gerenciadas facilmente através do painel de tarefas. A Figura 4 mostra o painel de adição para uma tarefa local. Nele podemos adicionar anotações, configurar a prioridade, marcar o status e agendar a tarefa para uma data futura.

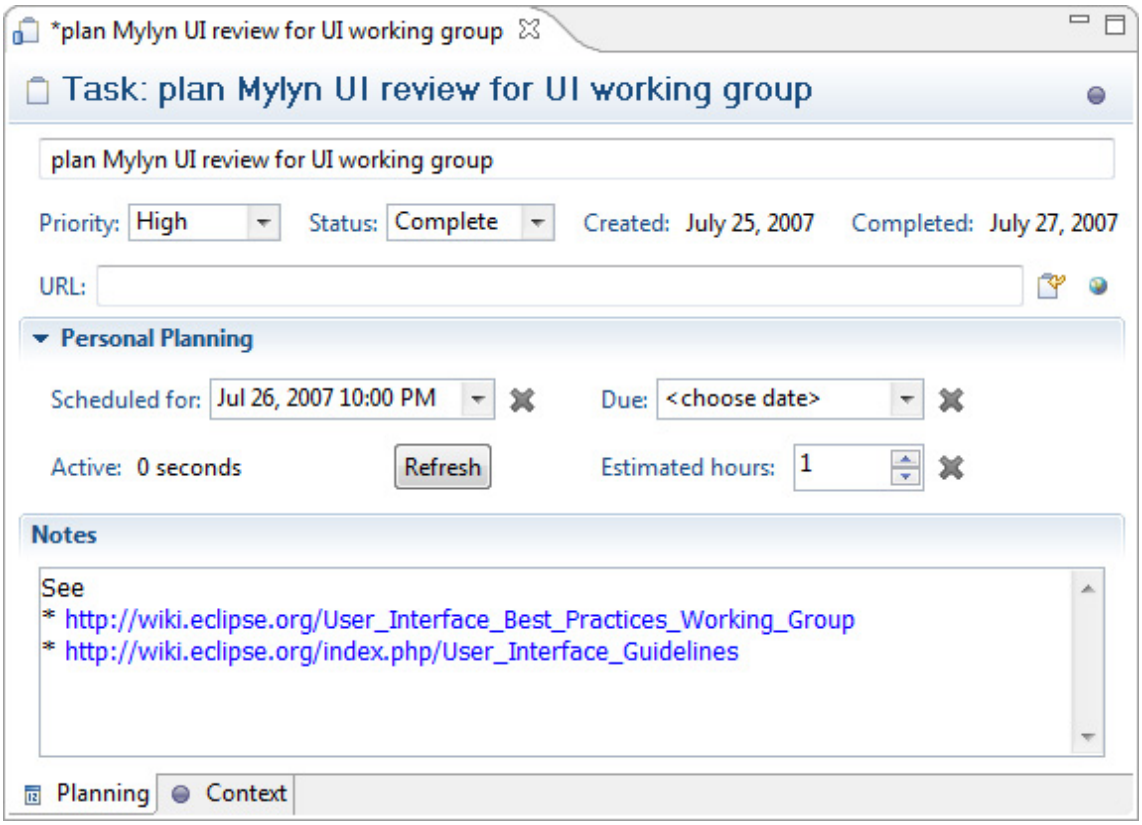

Figura 4: Adicionando uma tarefa local

2.4.2 Gestão de repositórios

O Mylyn introduz o conceito de conector para abstrair o acesso a um repositório de tarefas. Assim, é possível se integrar independentemente com uma vasta quantidade de repositórios e os conectores podem variar no nível de maturidade e integração. Os conectores para Bugzilla e Trac, incluídos por padrão no Mylyn, são considerados maduros e implementações de referência.

Um conector completamente integrado deve suprir os seguintes requisitos:

- **Consultas**: Consultas são o mecanismo para acessar conjuntos de tarefas armazenadas para popular a lista de tarefas do Mylyn. A edição de consultas é específica para cada conector.
- **Edição integrada**: Tarefas e consultas podem ser editadas com um editor integrado. Este recurso provê ligações para tarefas e outros elementos estruturados, assim como integração com o *desktop* e com o Eclipse como *drag-and-drop*.
- **Anexos**: Arquivos podem ser anexados e recuperados do repositório. Através deste recurso é possível compartilhar, por exemplo, o contexto de tarefas.
- **Modo offline**: Permite trabalhar desconectado e acessar tarefas e consultas imediatamente sem a necessidade de esperar pelo servidor. Este recurso também provê notificações de mudanças, que permite usar a lista de tarefas como caixa de entrada para tarefas.

A Figura 5 mostra os passos pare adicionar um repositório ao *workspace*. Primeiro é selecionado o tipo de repositório para determinar o conector a ser usado, em seguida, é provida a URL do repositório e a informação para autenticação, caso seja necessário.

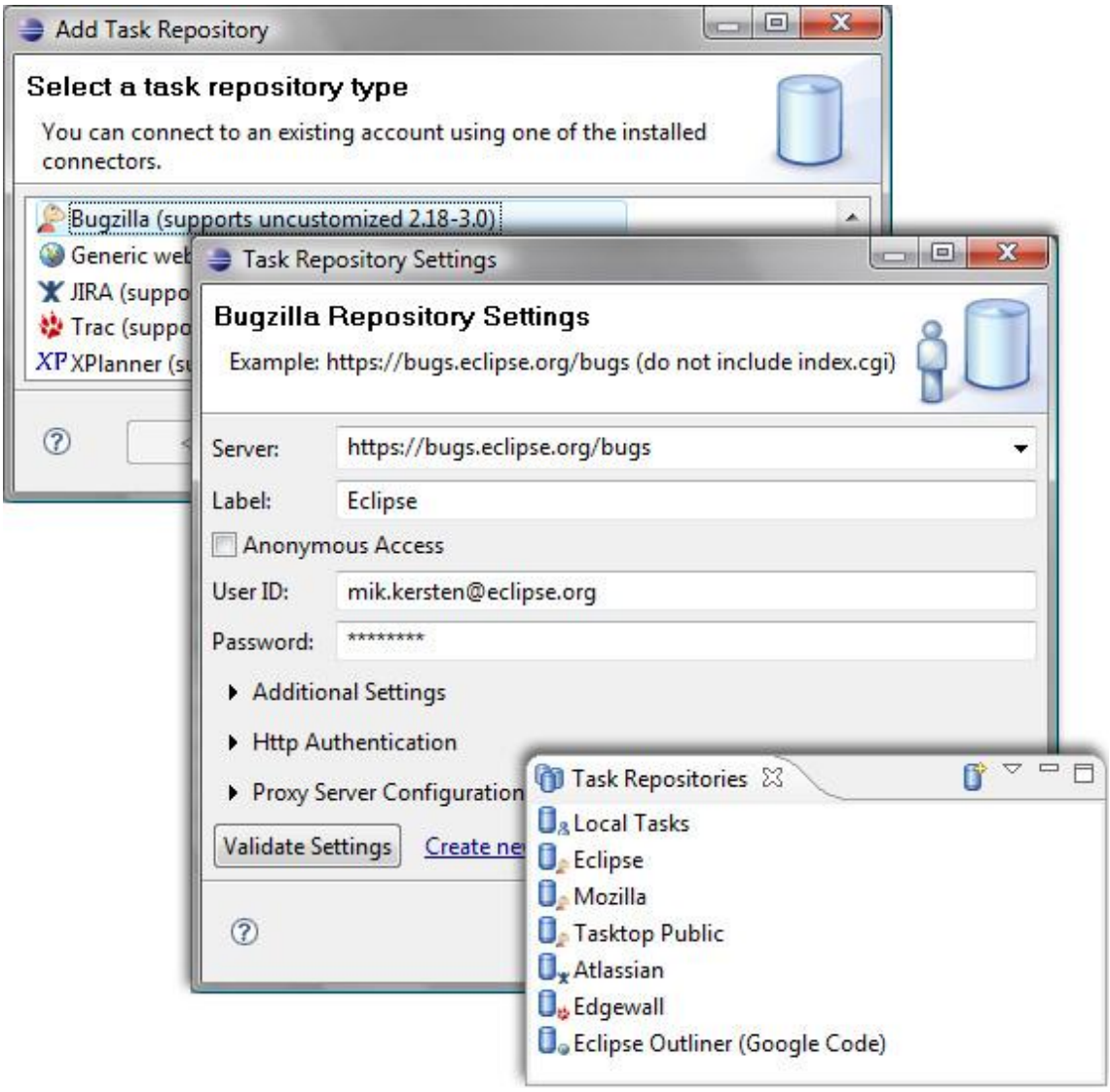

Figura 5: Adicionando um repositório de tarefas

2.4.3 Busca e filtragem

O Mylyn permite que se associem tarefas a determinadas categorias. É possível também criar filtros para a lista de tarefas. Desta forma ,evita-se ao máximo a sobrecarga de informação sobre o desenvolvedor. Além da categorização e da filtragem, o Mylyn permite a ordenação das tarefas por vários critérios e inclui uma busca rápida no painel de tarefas para sanar os casos onde é inevitável a sobrecarga. Todas essas características estão ilustradas na Figura 6.

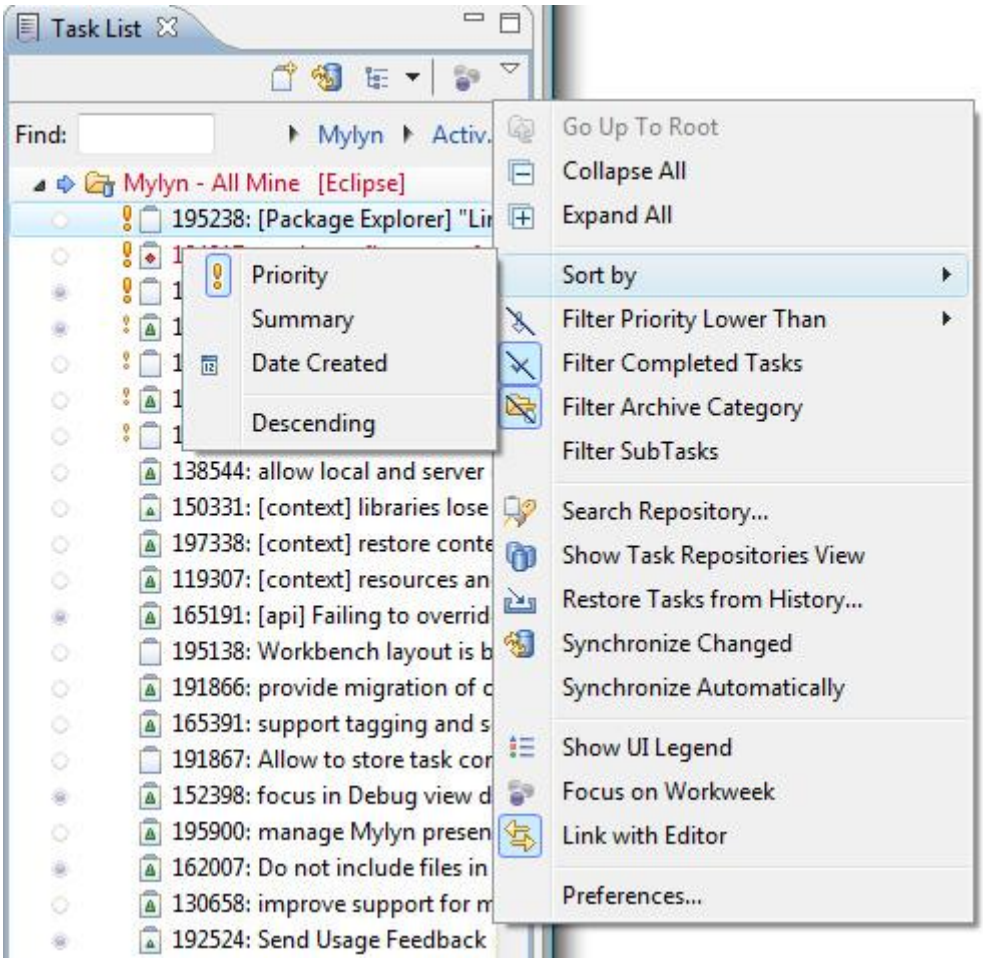

Figura 6: Recursos de busca, categorização, ordenação e filtragem

3 Visual Studio 2005

O Visual Studio 2005 é o principal produto de desenvolvimento de software da Microsoft. Ele baseia-se em uma IDE que permite a criação de aplicações *desktop*, *websites*, aplicações *web* e *web services* que rodam em qualquer plataforma suportada pelo .NET Framework. Plataformas suportadas incluem servidores e estações de trabalho Windows, dispositivos móveis e navegadores *web*.

Visual Studio 2005 está disponível em várias edições: Express, Standard, Professional, Tools for Office e um conjunto de cinco edições do Visual Studio Team System. As últimas são fornecidas em conjunção com assinaturas do MSDN Premium, cobrindo quatro dos principais papéis no desenvolvimento de software: Arquitetos, Desenvolvedores, Testadores, e Profissionais de Bancos de Dados. A funcionalidade combinada das quatro edições do Team System é oferecida em uma edição Team Suite (11).

3.1 Edições

Em suma, o Visual Studio 2005 está disponível no mercado em edições diferentes, cada uma mais adequada a um determinado público alvo (12):

- **Visual Studio 2005 Team System:** Edição mais completa, voltada para equipes maiores, visando prover uma solução integrada em Application Lifecycle Management[23]. Possui recursos avançados de colaboração envolvendo ferramentas, processos e orientação.
- **Visual Studio 2005 Professional Edition:** Esta é a edição mais completa voltada para desenvolvedores individuais. Inclui os recursos da edição Standard e algumas ferramentas adicionais.
- **Visual Studio 2005 Tools for the Microsoft Office System:** Versão especial do Visual Studio, com preço equivalente à edição Professional, destinada a desenvolvedores que estão construindo soluções compatíveis com o Microsoft Office[24] especificamente.
- **Visual Studio Standard Edition:** Edição básica para programadores profissionais. Inclui a linha completa das linguagens do Visual Studio (Visual Basic, C#, C++, e J#) e permite criar aplicações Windows, web e móveis.
- **Visual Studio Express Editions:** Edições (5 no total) com a finalidade de fornecer um ambiente de programação básico, acessível e fácil de aprender para estudantes, entusiastas, iniciantes e pessoas que programam por diversão.

3.2 Extensibilidade

[23] http://msdn2.microsoft.com/en-us/teamsystem/bb400737.aspx
[24] http://office.microsoft.com

Quando o desenvolvimento parece estar constantemente mudando enquanto as novas tecnologias, arquiteturas, e metodologias competem pela atenção do desenvolvedor uma coisa permanece constante: desenvolvedores são mais produtivos quando têm as ferramentas que se adéquam às suas necessidades. Tendo isso em vista, o Visual Studio permite que desenvolvedores construam extensões para ampliar suas capacidades. Essas extensões são anexadas ao ambiente Visual Studio oferecendo recursos que vão além dos que o próprio Visual Studio já oferece, como sistemas customizados de ajuda, ferramentas, novas linguagens e editores.

Nesta seção teremos uma visão geral dos recursos de extensibilidade disponíveis no Visual Studio 2005.

3.2.1 Níveis de extensão

O Visual Studio possui vários métodos de extensão, cada um variando no nível de poder dado ao desenvolvedor e funcionalidade, como pode ser visto na Figura 7.

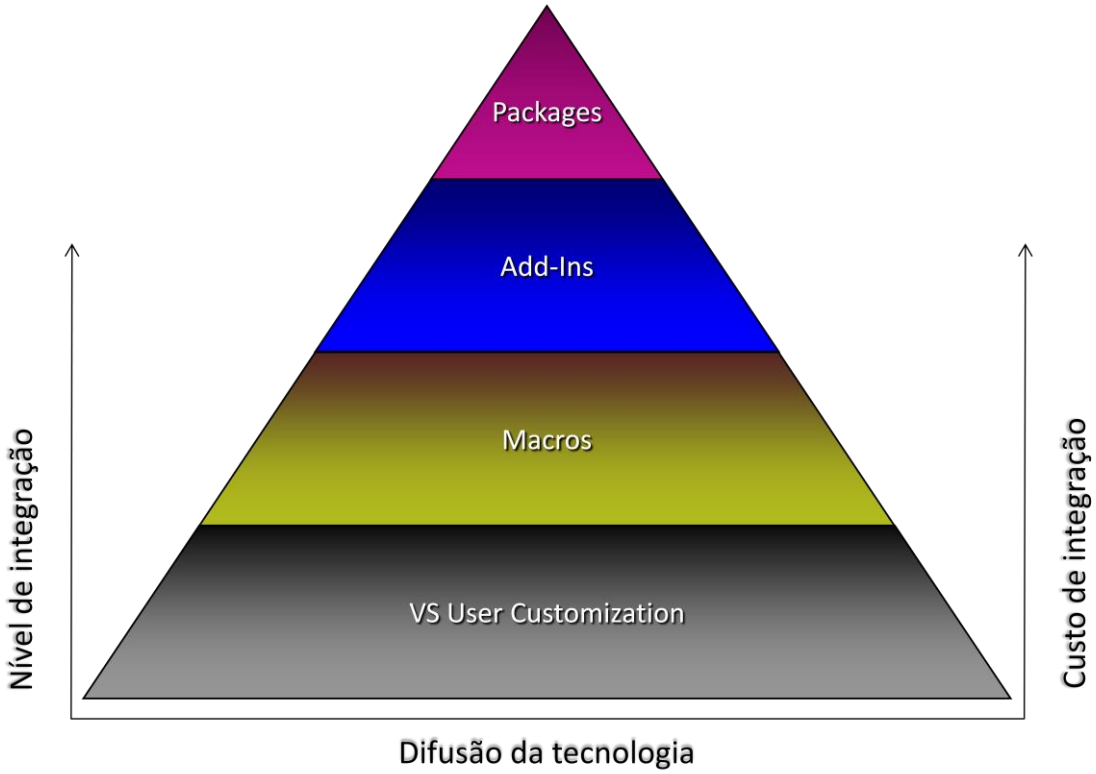

Figura 7: Opções para extender o Visual Studio 2005

Na base da pirâmide estão os mecanismos de customização do próprio Visual Studio, que permitem ao usuário configurar e compartilhar numa equipe os perfis de exibição da IDE de acordo com a finalidade, além das demais configurações relativas aos editores e ao comportamento padrão do ambiente. A Figura 8 mostra o diálogo que aparece ao executar a IDE pela primeira vez, onde podemos selecionar um perfil de exibição.

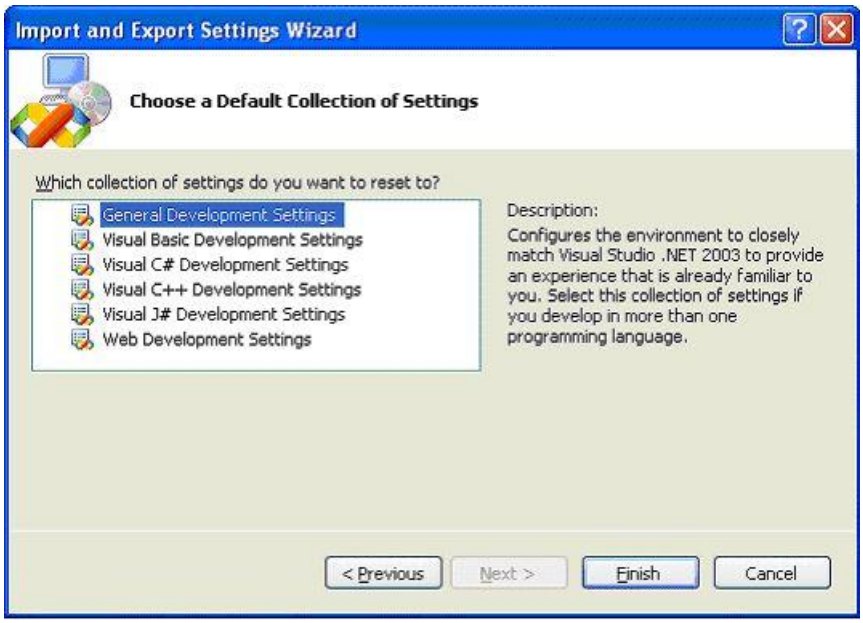

Figura 8: Diálogo para seleção de perfil

3.2.1.1 Macros

No nível seguinte estão os macros, a maneira mais simples de se adicionar um certo nível de automação ao Visual Studio. Os macros permitem a gravação e reprodução de comandos repetitivos com o intuito de aumentar a produtividade, eles são essencialmente scripts Visual Basic que podem ser editados acessando boa parte dos recursos do Visual Studio através do Automation Object Model[25].

3.2.1.2 Add-ins

Um nível acima dos macros estão os *add-ins*. Os *add-ins*, diferentemente dos macros, podem ser escritos em qualquer linguagem com suporte a COM[26], e proporcionam um controle total sobre o Automation Object Model do Visual Studio. Essa habilidade facilita a interação com a maior parte das ferramentas e funcionalidades do Visual Studio. Além disso, com *add-ins* é possível criar novas janelas de ferramentas, comandos e páginas de opções. Porém, *add-ins* não podem implementar novos tipos de documentos, de projetos, novos mecanismos de debug, etc. os quais requerem o próximo nível de extensão, os *packages*.

3.2.1.3 Packages

No topo da pirâmide estão os Packages, ou VsPackages. Este modelo de extensão do Visual Studio é o mais poderoso e mais complexo de todos. VsPackages fornecem opções de integração profundas e dão acesso para os afiliados do programa Visual Studio Industry Partners (VSIP) aos mesmos componentes e ferramentas utilizados pela Microsoft para criar o Visual Studio 2005.

---

[25] http://msdn2.microsoft.com/en-us/library/za2b25t3(vs.80)
[26] http://www.microsoft.com/com/

Um VsPackage é um pacote de software especializado, projetado para interagir com o Visual Studio no intuito de expor funcionalidades personalizadas. Extensões típicas do Visual Studio podem ser feitas como um único VsPackage ou como uma coleção de VsPackages que funcionam harmonicamente para expor uma funcionalidade relacionada. Cada VsPackage implementa uma interface especializada que permite que o Visual Studio crie, gerencie e organize os recursos associados com o pacote conforme o necessário. Adicionalmente, VsPackages também podem implementar a interface `IProfferService` para ofertar seus próprios serviços, o que permite ao pacote expor estado, lógica e funcionalidade desejados para outros VsPackages em maneiras que possibilitam integração e interação bastante poderosas.

VsPackages podem estender o Visual Studio oferecendo serviços, editores especializados, tipos de projetos personalizados, navegadores, janelas de ferramenta e outras ferramentas de desenvolvimento personalizadas. O ponto chave no que diz respeito aos VsPackages é que eles são unidades de funcionalidade completamente auto-contidas e atômicas, que podem ser distribuídas, invocadas e gerenciadas conforme o necessário.

4 Trac

O Trac é um sistema *open source* de *issue tracking* e wiki[27], voltado para projetos de desenvolvimento de software. Ele utiliza uma abordagem minimalista para gestão distribuída de projetos de software.

Outra característica fundamental do Trac é interligação entre os artefatos do projeto. Com ele é possível manter relacionamentos entre *bugs*, tarefas, *changesets*, arquivos e páginas wiki (13).

Dentre os muitos usuários[28] do Trac, estão o Ruby on Rails[29] e o Jet Propulsion Laboratory [30] da NASA (14).

4.1 Subsistema de tickets

O banco de dados de *tickets* do Trac (15) proporciona recursos para rastreamento de *bugs*, pendências e tarefas relacionadas ao projeto. Como elemento central de gestão do Trac, os *tickets* são usados para representar tarefas do projeto, requisições de mudanças, notificações de *bugs* e questões de suporte.

O fluxo de trabalho do Trac (Figura 9) é leve. Um ticket é atribuído a uma pessoa que deve resolvê-lo ou atribuí-lo a outro colaborador. Todos os tickets podem ser editados, anotados, priorizados e comentados a qualquer momento.

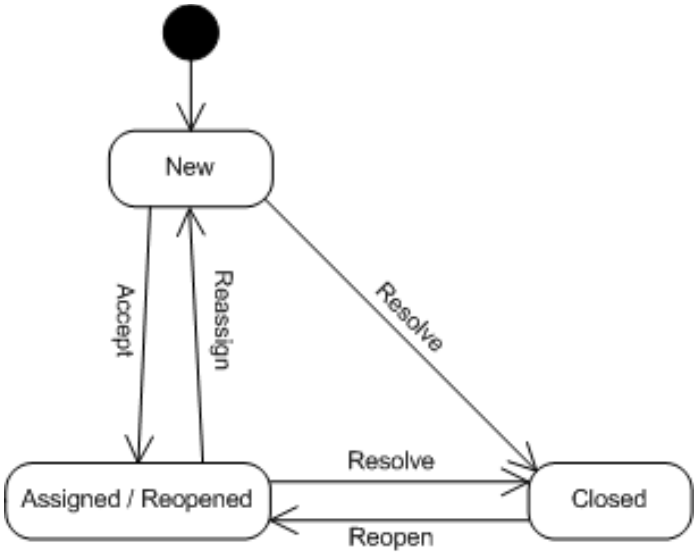

Figura 9: Diagrama de estados dos *tickets* no Trac

[27] http://wikipedia.org/wiki/Wiki
[28] http://trac.edgewall.org/wiki/TracUsers
[29] http://dev.rubyonrails.org/
[30] http://www.jpl.nasa.gov/

O Trac também possui um recurso chamado *Roadmap* (Figura 10), uma visão do sistema de tickets que auxilia o planejamento e a gestão do desenvolvimento futuro do projeto. Basicamente, o *Roadmap* é uma lista de *milestones* futuros (marcos), cada um contendo, por exemplo, uma descrição dos seus objetivos principais. Além disso, os *tickets* previstos para cada *milestone* são agregados, e a razão entre tickets resolvidos e em aberto é exibida na barra de progresso do *milestone*.

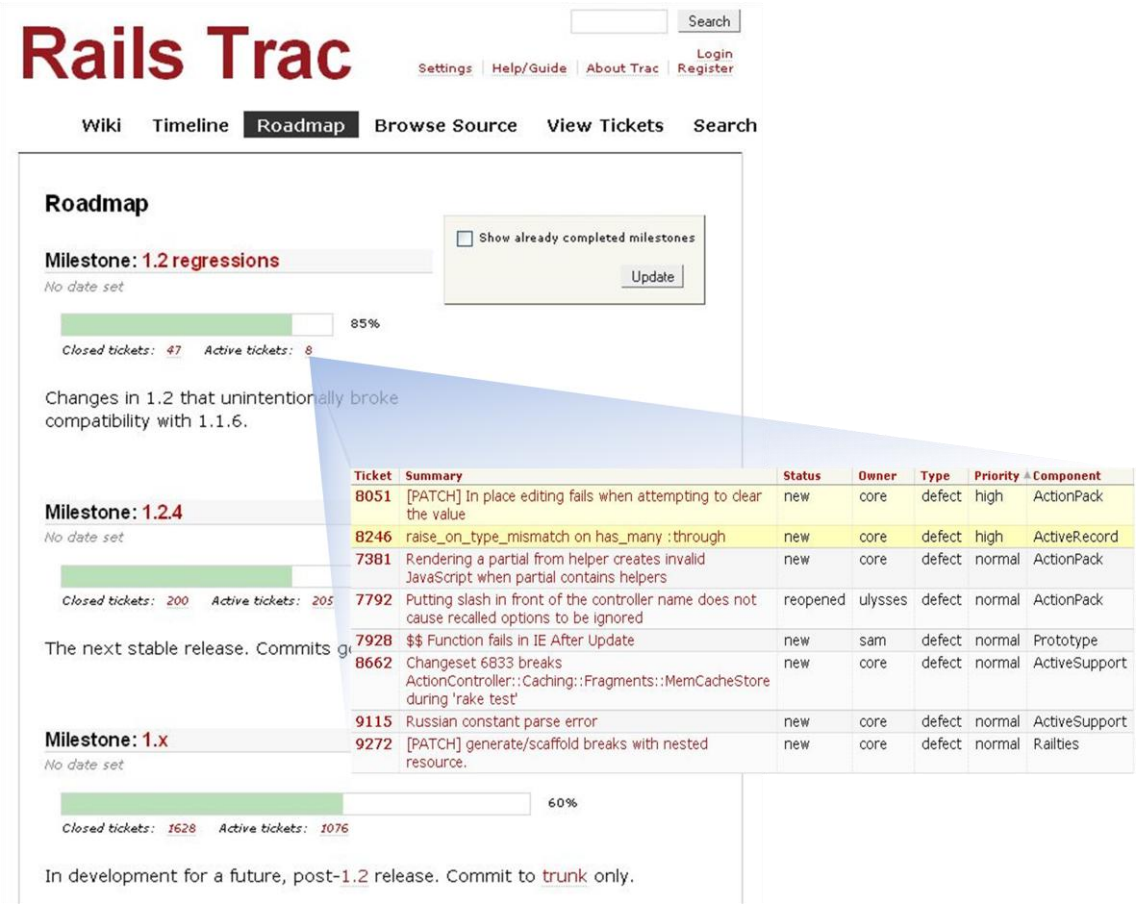

Figura 10: Trac Roadmap e Ticket Query

4.2 Subsistema de controle de versão

A forte integração entre o Trac e sistemas de controle de versão permite identificar as revisões e modificações que foram feitas para concluir uma determinada tarefa. A sintaxe do wiki utilizada no Trac permite links diretos para vários objetos. Por exemplo, "#21" é uma referência ao *ticket* Nº 21, e "[124]" refere-se ao *changeset* (ou revisão) Nº 124. É possível também navegar por diretórios e revisões específicas de arquivos armazenados no repositório através do *Repository Browser*.

4.3 Timeline

Na Figura 11 podemos ver um exemplo do *Timeline*, um recurso que permite monitorar de forma efetiva a atividade do projeto. Ele provê uma visão histórica do projeto listando os

eventos que ocorreram em ordem cronológica num único relatório. Cada item contém um link para o evento em questão.

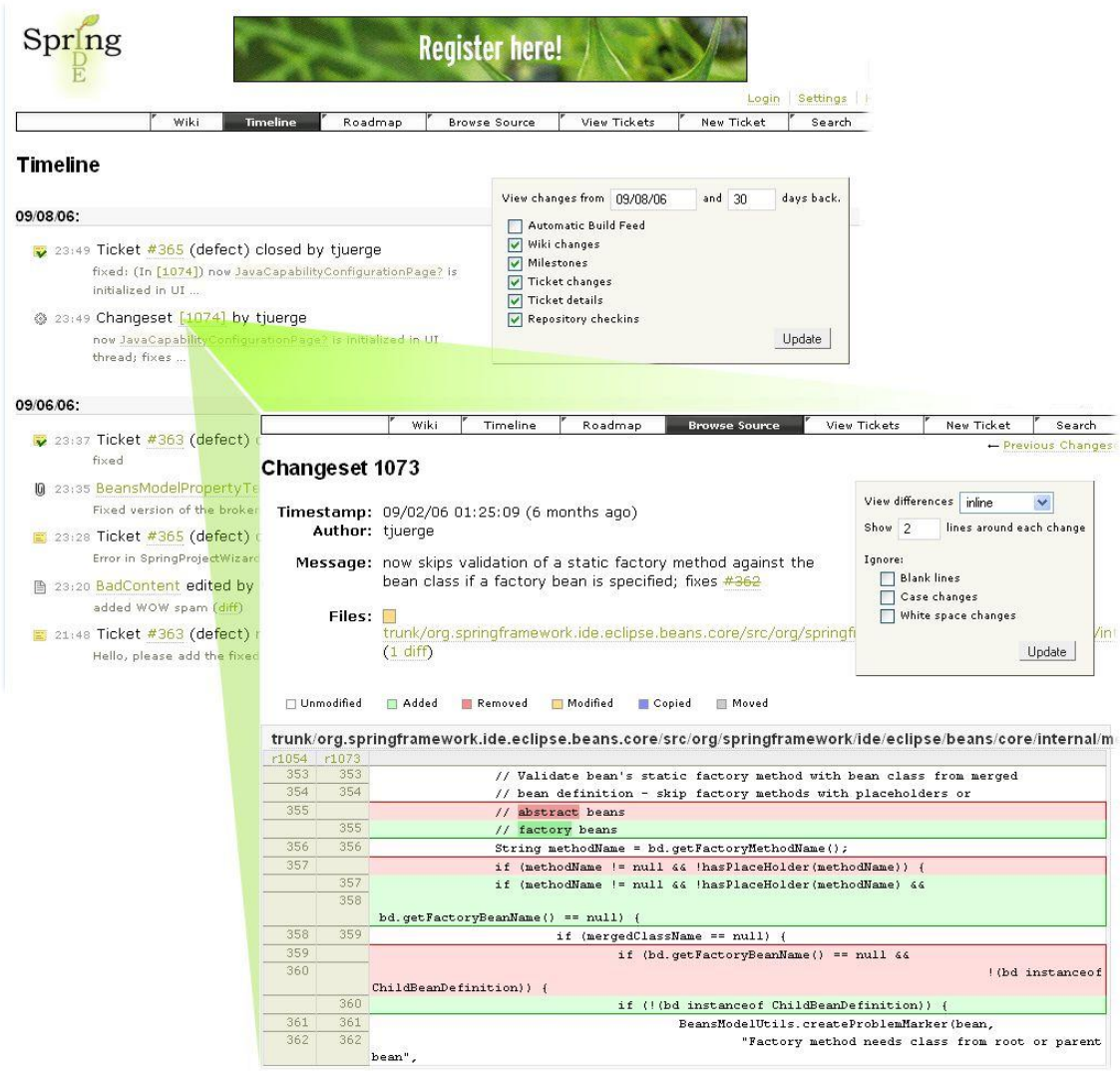

Figura 11: Trac Timeline e Changeset

4.4 Acesso externo

O Trac permite o acesso a partir de outras aplicações através de uma interface *web service*. Esta funcionalidade é oferecida através do Trac XML-RPC Plugin[31]. Como o próprio nome sugere, esta interface utiliza o protocolo XML-RPC [32], um protocolo aberto de interoperabilidade baseado em XML[33] sobre HTTP[34]. O Trac XML-RPC Plugin será incorporado à próxima versão do Trac, a 0.11.

[31] http://trac-hacks.org/wiki/XmlRpcPlugin
[32] http://www.xmlrpc.com/
[33] http://en.wikipedia.org/wiki/XML
[34] http://en.wikipedia.org/wiki/HTTP

5 VSFocus

O processo de desenvolvimento do VSFocus envolveu muitas decisões entre estratégias, ferramentas e tecnologias utilizadas, com o intuito de dar um bom pontapé inicial na produção de um conjunto de ferramentas que possa trazer as vantagens e recursos apresentados pelo Mylyn para outros ambientes de desenvolvimento além do Eclipse, no caso deste trabalho de graduação, o Visual Studio. Neste capítulo serão discutidas essas decisões tomadas no decorrer do desenvolvimento do VSFocus, além dos resultados obtidos com o trabalho.

5.1 Público alvo

Antes de falar explicitamente sobre os requisitos levantados é importante definirmos, mesmo que brevemente, o público a que se destinaria o VSFocus. Considerando suas características como um produto já finalizado, com recursos semelhantes aos do Mylyn. Para isso, foi determinado um cenário, com a visão de uma situação típica do mercado de software, que exemplifica uma parte do possível público para o VSFocus.

5.1.1 Cenário

Considere uma fábrica de software de médio porte que possui um conjunto de processos definidos para todos os seus projetos em andamento, por exemplo, os processos relativos a gerência de configuração estabelecidos pelo CMMI [35] nível 2. Para implementar esses processos, a fábrica integra uma base de ferramentas de suporte ao desenvolvimento, digamos: um *issue tracker*, um sistema de controle de versão, IDE, *templates*, geradores de relatórios, etc.

Então, a fábrica recebe um novo projeto que precisa ser desenvolvido utilizando a linguagem de programação C++ [36] num ambiente Windows. Pela liderança de mercado e pela familiaridade de boa parte dos desenvolvedores da fábrica, a IDE de escolha é o Visual C++ 2005[37].

O custo de migração para o ambiente VSTS é muito alto, pois além da aquisição das licenças seria necessário substituir as ferramentas de suporte para a solução ALM do VSTS. Isso traria também um impacto muito grande para os outros projetos, que não estão baseados na plataforma Visual Studio. Logo, o VSFocus neste caso surge como uma solução interessante, dando a possibilidade de se utilizar uma versão mais adequada do Visual Studio, como a Professional (vide seção 3.1) e facilitando, pela integração, a aderência aos processos existentes.

---

[35] http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration
[36] http://en.wikipedia.org/wiki/C++
[37] http://msdn.microsoft.com/visualc/

5.2 Requisitos elicitados

Vimos no capítulo 2 que o Mylyn, apesar de ser um projeto relativamente novo, já apresenta um vasto conjunto de recursos e vem crescendo bastante tanto em tamanho quanto em adoção pela comunidade de usuários do Eclipse[38]. Portanto, foi necessário escolher dentre esses recursos os que apresentassem a maior relevância possível para a comunidade, que dessem a maior margem para explorar as possibilidades de extensão do Visual Studio, e que fossem viáveis de implementação em tempo hábil no contexto deste trabalho.

Com base nos critérios citados acima, a ferramenta Mylyn foi analisada de acordo com seus 3 *frameworks* básicos, apresentados na seção 2.3. Tanto o primeiro quanto o segundo, *Monitor* e *Context*, se mostraram excessivamente complexos no que diz respeito ao modelo e ao nível exigido de integração com a IDE. Já o terceiro, *Tasks*, pareceu bem mais palpável nesses quesitos, além de apresentar benefícios mais visíveis e imediatos a princípio, caso fosse possível a integração com algum *issue tracker*.

5.2.1 Requisitos não-funcionais e restrições tecnológicas

Após a seleção do *framework* a ser explorado foi necessário validar a escolha da IDE e concluir a escolha do primeiro *issue tracker* que seria contemplado no escopo do projeto. Para a escolha da IDE, além da motivação apresentada no capítulo 1, vimos na seção 3.2 que o Visual Studio 2005 possui capacidades de extensão suficientes para a implementação dos componentes de gestão de tarefas necessários, validando assim sua escolha.

Já para o *issue tracker* foi preciso analisar, dentre os citados na proposta inicial, as características mais relevantes para o contexto do projeto como: estabilidade, disponibilização de documentação, APIs para acesso externo, licença e base de usuários. Considerando esses critérios, concluiu-se que o Trac seria o sistema mais adequado no momento. O Trac, assim como o Mylyn, é um projeto relativamente novo comparado a sistemas mais tradicionais como o Bugzilla, mas vem se tornando um padrão *de facto,* principalmente em ambientes *open source*. Além disso, o Trac está licenciado sob uma licença BSD e permite um acesso externo simplificado e razoavelmente bem documentado através de uma interface em forma de *web service* utilizando o protocolo XML-RPC (seção 4.4).

5.2.2 Requisitos funcionais

Com base então no *framework Tasks* do Mylyn, foi possível chegar a um conjunto inicial de requisitos funcionais para o VSFocus. Detalhando o que foi mencionado na seção 1.1 temos:

---

[38] Durante o desenvolvimento deste trabalho de graduação o Mylyn foi promovido da categoria de projeto de pesquisa (*Technology*) para a categoria de ferramenta oficial do Eclipse (*Tools project*, http://www.eclipse.org/projects/) e tornou-se um componente incluso por padrão na versão mais recente da IDE, o Eclipse Europa, lançada no fim de junho de 2007 (http://www.eclipse.org/org/press-release/20070627_europarelease.php).

5.2.2.1 Gestão de repositórios

O VSFocus deve incluir uma maneira de configurar acesso a repositórios Trac, de maneira semelhante ao que foi apresentado na seção 2.4.2.

5.2.2.2 Painel de tarefas

Deve também ser criado um painel de tarefas, de preferência integrado ao já existente no Visual Studio 2005, que exiba uma lista das tarefas atribuídas ao usuário configurado para cada um dos repositórios definidos. Caso um dos repositórios tenha acesso anônimo, todas as tarefas do mesmo devem ser carregadas.

5.2.2.3 Busca e filtragem (desejável)

Os recursos do Mylyn apresentados na seção 2.4.3 também poderiam ser portados para o VSFocus.

5.2.2.4 Suporte ao Trac

Como mencionado na seção anterior (5.2.1), o Trac foi selecionado como o *issue tracker* de implementação prioritária. Portanto, o VSFocus deve dispor de um conector para Trac parcialmente integrado, que disponha de um mecanismo de consultas (vide seção 2.4.2).

5.2.2.5 Suporte a outros issue trackers (desejável)

Se possível, o VSFocus pode implementar conectores para outros repositórios além do Trac. Tipos de repositórios alternativos incluem o Bugzilla, o *issue tracker* do Google Code Hosting, o Mantis, e o Atlassian JIRA.

5.3 Ambiente de desenvolvimento

O VSFocus está hospedado no site Assembla.com[39], que disponibiliza todos os recursos de colaboração e gestão necessários (Subversion, Trac, wiki, blog, páginas personalizadas, etc.) e foi desenvolvido com as seguintes versões das ferramentas:

- Visual Studio 2005 Team Suite
- Visual Studio 2005 SDK Version 4.0 (2007.02)
- Trac 0.10.4 com Trac XML-RPC Plugin 0.1
- XML-RPC.NET 2.1.0
- Subversion 1.4.3

5.4 Arquitetura e Implementação

Para implementar os requisitos descritos na seção anterior mantendo uma arquitetura extensível o suficiente para futuros desenvolvimentos, foi necessário tomar uma abordagem semelhante à do Mylyn, desacoplando ao máximo os elementos de modelo independentes e os elementos responsáveis pela integração com a IDE e pelo controle e construção da interface

[39] http://www.assembla.com/

gráfica. Além disso, o nível de integração exigido pelos requisitos apresentados também influenciou na escolha do modelo de extensão do Visual Studio. O requisito definido na seção 5.2.2.2 em especial demandou que fosse feita uma opção pelo método de Packages (seção 3.2.1.3), o que fez aumentar consideravelmente o nível de complexidade do projeto.

5.4.1 Módulos

O VSFocus está disposto em dois projetos separados: `VSFocus` e `VSFocusTracConnector`. O primeiro é o VsPackage que trata da integração com o Visual Studio e contém os módulos relativos à interface gráfica para a gestão de tarefas, o segundo é uma biblioteca de classes que implementa o conector para o Trac.

Uma visão geral dos principais componentes do `VSFocus` é apresentada na Figura 12. Na camada de controle podemos ver o `RepositoryManagerControl`, responsável pelo controle dos componentes que formam a interface para a gestão de repositórios, representados pelo RepositoryManagerPane e pelo AddRepositoryDialog. No mesmo diagrama está o `RepositoryTaskProvider`, que é responsável pela sincronização da lista de tarefas. Esse componente utiliza a biblioteca do projeto `VSFocusTracConnector` para buscar as tarefas associadas ao usuário.

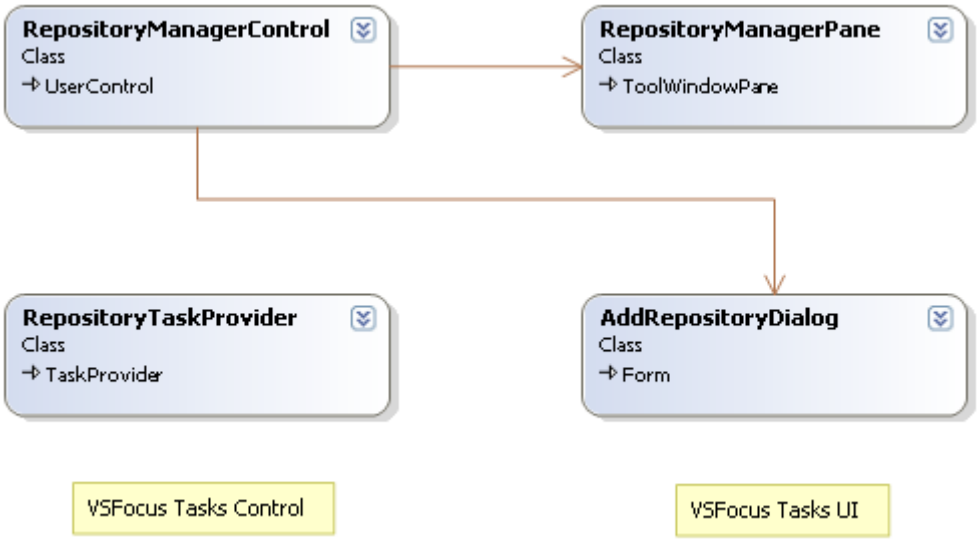

Figura 12: Módulos do VSFocus

Os diagramas da Figura 13 e da Figura 14 apresentam os principais componentes presentes no projeto `VSFocusTracConnector`. A interface ITracClient é uma abstração para um conector para Trac. Isso permite que possam ser feitas diferentes implementações, com métodos de acesso a repositório diferentes. A interface `TracXmlRpcClientProxy` funciona como um Proxy para o *web service* do Trac a ser acessado. Esse componente é a peça-chave para a integração com a biblioteca XML-RPC.NET. Através dele, a implementação do cliente `TracXmlRpcClient` pode fazer chamadas remotas transparentemente, como se fossem locais. Na Figura 14 estão representadas algumas das classes de dados referentes à

camada de modelo da biblioteca. Essas são implementações específicas para o Trac. Por exemplo, a classe `TracTicket` representa o conceito de um *ticket*, através dela é que os dados são transportados entre o cliente e o servidor da aplicação.

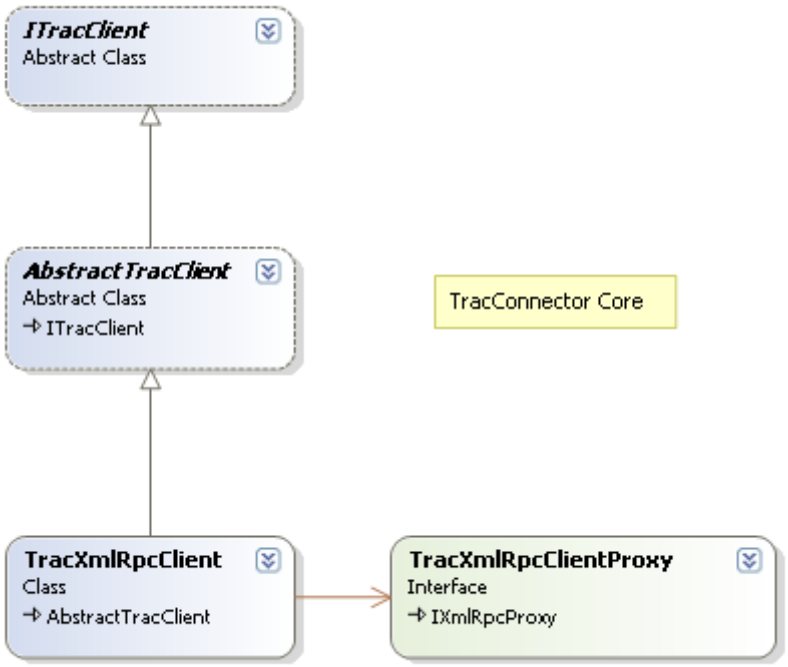

Figura 13: Componentes do VSFocus (core)

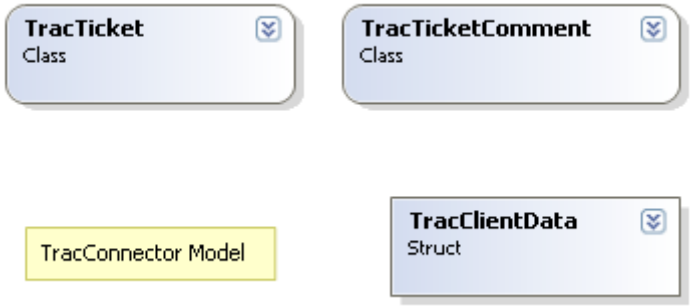

Figura 14: Componentes do VSFocus (model)

5.5 Experiência do usuário

Para dar uma idéia melhor do resultado obtido na implementação, serão apresentadas nesta seção algumas capturas de tela do VSFocus em execução.

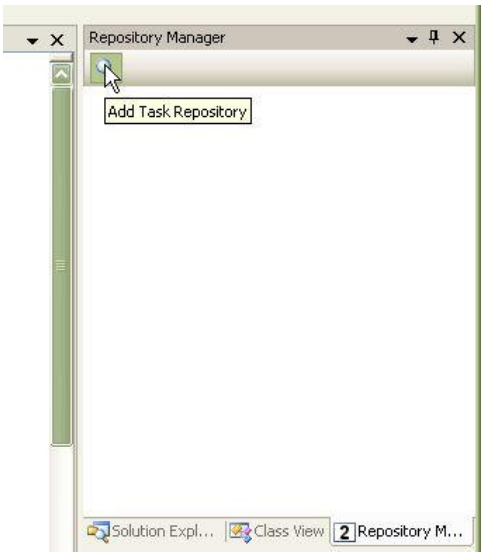

Figura 15: Tool Window do gerenciador de repositórios

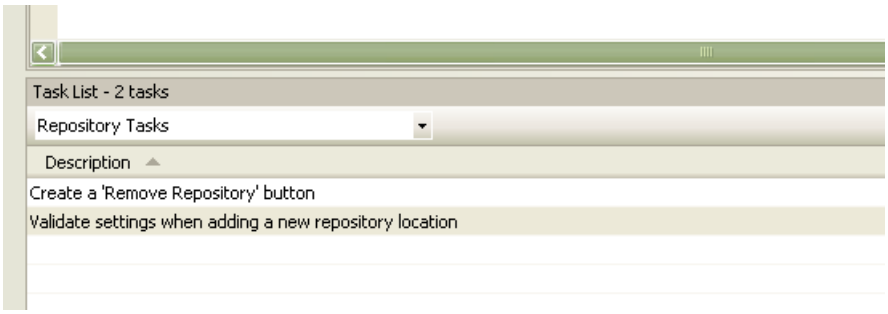

Figura 16: Integração da lista de tarefas

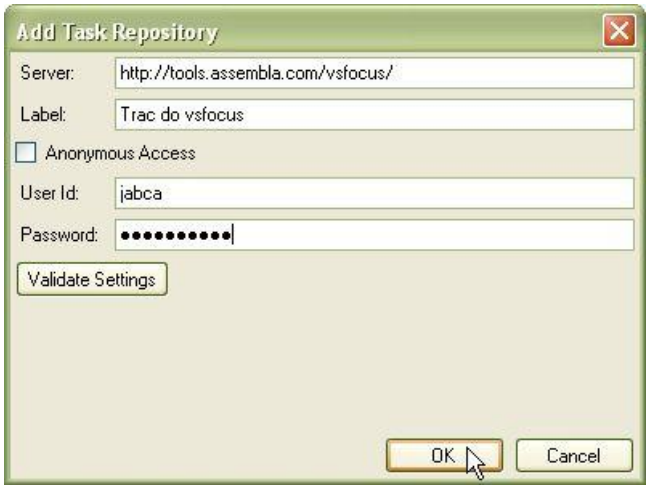

Figura 17: Adicionando um novo repositório

6 Considerações Finais

Apesar de todas as dificuldades encontradas durante a realização deste trabalho, ele foi de grande valia, no mínimo, para a formação e aprendizado do aluno. Espera-se que seu valor seja reconhecido e usufruído também por toda a comunidade acadêmica e de desenvolvedores.

Devido ao caráter multidisciplinar do projeto, muitos conhecimentos foram adquiridos e aplicados satisfatoriamente. A área de gerência de configuração foi bem explorada com a pesquisa sobre rastreamento de requisitos e integração de ferramentas. O modelo de extensibilidade do Visual Studio 2005 também representou uma parte importante do campo de conhecimento abordado no projeto. O estudo a respeito de uma ferramenta inovadora como o Mylyn, certamente contribuiu também para uma expansão considerável de horizontes no que diz respeito ao que temos hoje e ao que poderemos ter no futuro dos ambientes integrados de desenvolvimento.

6.1 Dificuldades encontradas

Muitas barreiras foram encontradas no decorrer do desenvolvimento do VSFocus, dentre elas, podemos citar aqui as dificuldades que mais afetaram o resultado final obtido no projeto.

- Inexperiência com a plataforma Visual Studio e com a linguagem C#: Muito esforço foi despendido neste quesito. Além de ter sido necessário um alto nível de integração com a IDE, que conseqüentemente acarretou uma maior complexidade para o projeto, foi necessário o aprendizado de uma linguagem de programação praticamente nova para o aluno.
- Caráter inovador das tecnologias abordadas no projeto: muitas das tecnologias utilizadas para o desenvolvimento do VSFocus estavam, no momento, em estados não muito maduros de desenvolvimento, como o Mylyn e o próprio Visual Studio SDK.
- Isolamento acadêmico: o trabalho foi proposto completamente fora de um contexto de pesquisa maior, como um outro projeto de escopo mais abrangente desenvolvido na mesma instituição.
- Restrições de tempo: a proposta do projeto sofreu muitas modificações em sua fase inicial, o que ocasionou algumas reduções no escopo.

6.2 Trabalhos futuros

Visto que este projeto toma o Mylyn como inspiração e apresenta o protótipo de um pequeno conjunto do que pode ser explorado em outras IDEs além do Eclipse, podemos certamente concluir que ainda há muito o que ser feito, e destacar o que de mais interessante pode ser considerado para trabalhos futuros.

Em primeiro lugar, o mais óbvio, a continuação do projeto VSFocus abrangendo os requisitos desejáveis não implementados neste protótipo inicial, incluindo a implementação de conectores para outros sistemas de gestão de tarefas, melhorias na interface e na integração

com o Visual Studio. Nesse contexto também podemos incluir a implementação dos demais *frameworks* apresentados pelo Mylyn: o *Monitor* e o *Context*.

Levando em conta também a aplicação dos conceitos de orientação a tarefas vindos do Mylyn, é plausível que se sugira iniciar projetos semelhantes com foco em outras IDEs populares, além do Eclipse e do Visual Studio.

Outra necessidade crucial para a expansão do VSFocus entre a comunidade é a criação de uma documentação técnica bem elaborada e concisa para a sua base de código. Sem isso é praticamente impossível conquistar a participação de terceiros no projeto.

7 Referências Bibliográficas

1. **Kersten, Mik.** Mylyn 2.0, Part 1: Integrated task management. *IBM developerWorks.* [Online] 14 de Agosto de 2007. [Citado em: 21 de Agosto de 2007.] http://www.ibm.com/developerworks/java/library/j-mylyn1/.

2. **Berczuk, Steve e Appleton, Brad.** *Software Configuration Management Patterns: Effective Teamwork, Practical Integration.* s.l. : Addison-Wesley, 2003. 0201741172.

3. **Spolsky, Joel.** Human Task Switches Considered Harmful. *Joel on Software.* [Online] 12 de Fevereiro de 2001. http://www.joelonsoftware.com/articles/fog0000000022.html.

4. **Coté, Michael.** More on Mylar: Tasks, Picking, Agile ALM. *People Over Process.* [Online] 12 de Junho de 2006. http://redmonk.com/cote/2006/12/06/more-on-mylar-tasks-picking-agile-alm/.

5. **Evans Data Corp.** *Developers' Choice IDE Scorecard.* 2006.

6. **Gunderloy, Mike.** Three must-have tools: Version control, issue tracking, unit testing. [Online] 23 de Maio de 2005. http://whatis.techtarget.com/tip/0,289483,sid8_gci1091076,00.html.

7. **Kersten, Mik.** Mylyn 2.0, Part 2: Automated context management. *IBM developerWorks.* [Online] 14 de Agosto de 2007. [Citado em: 21 de Agosto de 2007.] http://www.ibm.com/developerworks/java/library/j-mylyn2/.

8. Mylyn Homepage. *Site do Projeto Mylyn.* [Online] [Citado em: 14 de 8 de 2007.] http://www.eclipse.org/mylyn/.

9. Mylyn User Guide. *Mylyn Project Wiki.* [Online] [Citado em: 3 de 7 de 2007.] http://wiki.eclipse.org/Mylyn_User_Guide.

10. **Kersten, Mik.** *Focusing knowledge work with task context.* 2007.

11. Microsoft Visual Studio. *Wikipedia.* [Online] [Citado em: 7 de 7 de 2007.] http://en.wikipedia.org/wiki/Visual_studio.

12. **Gunderloy, Mike.** How-To-Select Your Best Licensing Option for Visual Studio 2005. *How-To-Select Guides.* [Online] 15 de 8 de 2005. [Citado em: 14 de 8 de 2007.] http://www.howtoselectguides.com/dotnet/visualstudio2005/.

13. **Smart, John Ferguson.** What issue tracking system is best for you? A review of Bugzilla, Trac, and JIRA. *JavaWorld.com.* [Online] 14 de 3 de 2007. [Citado em: 20 de 7 de 2007.] http://www.javaworld.com/javaworld/jw-03-2007/jw-03-bugs.html.

14. Who Uses Trac? *Site do Trac.* [Online] [Citado em: 4 de 8 de 2007.] http://trac.edgewall.org/wiki/TracUsers.

15. Trac User and Administration Guide. *Site do Trac.* [Online] [Citado em: 4 de 8 de 2007.] http://trac.edgewall.org/wiki/TracGuide.

16. Visual Studio Extensibility Overview. *Microsoft Developer Network.* [Online] [Citado em: 20 de 7 de 2007.] http://msdn.microsoft.com/vstudio/extend/vseoverview.

Data e assinaturas

Este Trabalho de Graduação é resultado dos esforços do aluno João Augusto de Britto Cavalcanti Alves, sob a orientação do professor André Luis Medeiros dos Santos e co-orientação de Ryan Leite Albuquerque, na participação do projeto "VSFocus: Orientação a Tarefas para Microsoft Visual Studio", conduzido pelo Centro de Informática da Universidade Federal de Pernambuco. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

Recife, 22 de agosto de 2007

André Luis Medeiros dos Santos (Orientador)

João Augusto de Britto Cavalcanti Alves (Proponente)

Ryan Leite Albuquerque (Co-orientador)