Matéria: MATA 26 - Tópicos Especiais em Sistemas Computacionais IV

Equipe: Luciana Lins, Mário Jorge, Rafael Glauber e Sérgio Gramacho

Seleção de Projeto OSS para Contribuição

I. Descrição do Projeto

1. Projeto:

Adium

2. Website:

http://adium.im/

3. Descrição:

Adium é um Software Livre que oferece um mensageiro instantâneo para diversos protocolos na plataforma Mac OS X.

II. Reconhecimento inicial

4. Há quanto tempo o projeto existe? Descreva brevemente o histórico do projeto.

O projeto tem mais de 10 anos de existência e apresenta o histórico descrito a seguir.

O Adium foi criado pelo estudante universitário Adam Iser e a primeira versão "Adium 1.0” foi lançada em setembro de 2001, suportando apenas o AIM (AOL Instant Messenger, lançado em 1997), tendo sido desenvolvido em cocoaime e usando a biblioteca TOC. Os números de versão do Adium, desde então, têm seguido um padrão

um tanto incomum. Existiram, nessa época, vários *upgrades* para Adium 1.0, terminando com o Adium 1.6.2c.

Em 2002, Evan Schoenberg e um número significativo de membros da FunMac Fóruns passaram a fazer parte do projeto, produzindo os lançamentos comunitários para o Adium. As melhorias contínuas sugeridas culminaram no lançamento da versão 1.6.5.05.

Neste ponto, a equipe Adium começou uma reescrita completa do código e em 2003, a biblioteca do Pidgin (anteriormente "Gaim"), libpurple (então chamado de "libgaim"), foi implementada por Colin Barrett, com a ajuda instrumental de Hammond da equipe Gaim, Scott Lamb e Evan transformando-o em uma biblioteca compatível com o Adium, com a finalidade de adicionar suporte para protocolos de mensagens instantâneas diferentes do AIM. A equipe Adium liberou essas mudanças na versão "Adium 2.0". Em 28 de dezembro de 2003 o Adium tornou-se oficialmente multiprotocolo.

No entanto, em 2004, Adium 2.0 acabou sendo renomeado para "Adium X" e lançou a versão 0.50, sendo considerada a metade da versão 1.0. Cerca de 10.000 pessoas chegaram a baixar e usar essa versão.

Adium X 0.88 foi a primeira versão compilada como um binário universal, permitindo que ele fosse executado nativamente não somente em Macs de plataforma Power, mas também em Macs baseados em plataforma Intel.

Em 2005, o trabalho da versão 1.0 se inicia com grandes mudanças arquiteturais, enquanto o desenvolvimento da série 0.xx continua simultaneamente.

Em 2006, a versão X 0.89.1 é liberada e atinge mais de um milhão de downloads.

Em 2007 a equipe finalmente decidiu mudar o nome para "Adium" e, como tal, a versão "Adium 1.0" foi lançada em 02 de fevereiro de 2007, após um ano e meio de esforço de todos os colaboradores. Devido a esse histórico, o Adium já possuiu duas versões "1.0", porém distintas e separadas no tempo por quase 6 anos.

5. Qual a licença do projeto?

A licença de uso é GNU GPL.

6. Qual o tamanho do projeto? Número de linhas de código? Número de classes?

Em análise feita no código do projeto e através de informações colhidas no website “ohloh” em 09/04/2012, a equipe chegou às seguintes informações sobre o projeto:

- Linhas de Código: 159.358 linhas de código;
- Classes: 644 classes próprias e usa 605 classes de APIs de sistema;
- Protocolos: 78 protocolos próprios e usa 113 protocolos de APIs de sistema;
- Funções: 4526 funções próprias e usa 9855 funções de APIs de sistema;
- Estruturas de dados: 484 structs próprios e 2192 structs de sistema;
- Unions: 21 unions próprios e 122 unions de sistema;
- Enums: 327 enums próprios e 3347 enums de sistema;
- Types: 829 types próprios e 5426 types de sistema;
- Globals (static): 257 globals próprios e 4550 globals de sistema.

7. O tamanho do projeto tem crescido nas últimas versões?

Segundo o *website* “ohloh” o projeto “tem crescido ano a ano, apresentando código estável e bem documentado, contando com a colaboração de um time grande e ativo”.

8. Existe atividade recente no projeto?

Sim. O *commit* mais recente foi há 5 horas antes da nossa coleta de dados e há 18 dias atrás houve um *commit* de mais de 1000 linhas.

9. Existe colaboração de empresas para o projeto? Existem mais desenvolvedores independentes do que funcionários de empresas?

Observamos que a grande maioria das contribuições de código não adveio de empresas, mas sim dos colaboradores independentes. Contudo, identificamos referências a contribuições de empresas para o projeto, como segue:

Recursos Doados:

A. NetworkRedux: NetworkRedux doou e hospeda um servidor dedicado, de alta velocidade que operacionaliza adium.im, adiumxtras.com, e caixas de correio associadas e listas de discussão. O website adium.im serve quase 3 milhões de page views por mês. Além disso, a empresa hospeda o sistema de controle de erros e de documentação AdiumTrac e o repositório do sistema de controle de versões do Adium baseado em tecnologia mercurial. NetworkRedux também deu apoio pessoal para os desenvolvedores do Adium.

B. CacheFly: CacheFly hospeda o download dos binários do Adium, servindo vários terabytes de dados mensais.

C. TheCodingMonkeys: TheCodingMonkeys doou várias licenças do editor colaborativo de textos SubEthaEdit, para uso pelos desenvolvedores do Adium. Esses editores são muito utilizados pelos colaboradores, fornecendo changelogs gerados para cada versão.

D. YourKit: doou um número de licenças da ferramenta CPU profiling, YourKit Java Profiler, que ajudou a otimizar o Adium.

E. Skorpiostech, Inc.: Ian Baird da Skorpiostech foi gentil o suficiente para doar várias cópias do Changes.app, um aplicativo de comparação de código, que mostrou-se muito útil para os colaboradores.

F. Penguin Militia Networks: há mais de dois anos, Penguin Militia Networks tem doado hospedagem no Trac e Subversion para o projeto Adium.

Contribuições de Empresas

G. Google: a Google convidou os representantes do projeto Adium para participar da organização do Google Summer of Code, em 2006, 2007 e 2008. A cada ano de participação, vários estudantes juntaram-se à equipe de colaboradores do projeto, o que ocasionou contribuições significativas para o desenvolvimento.

10. Quais são as tecnologias (linguagem, bibliotecas, bancos de dados) utilizadas pelo software?

O projeto está desenvolvido na linguagem Objective C, usando a biblioteca libpurple.

III. Identificação de Tarefas

11. Onde fica o repositório de código fonte? Qual ferramenta de controle de versão é utilizada?

O repositório de código fonte fica em http://hg.adium.im/adium e usa a tecnologia Mercurial de repositório distribuído.

12. Onde fica o repositório de bugs?

O repositório de *bugs* utiliza a ferramenta "TRAC" e está localizado em http://trac.adium.im/

13. Existe documentação para ajudar novos colaboradores? Que tipo de informação está disponível?

Sim, existe documentação que ajuda tanto a novos colaboradores (localizado em http://trac.adium.im/wiki/Development) quanto usuários da aplicação. Abaixo listamos algumas opções de documentação disponibilizadas na página do projeto:

A. Screencasts: vídeos contendo informações básicas de instalação, first run, adição de contatos, primeira troca de mensagens instantâneas e chatting.

B. Documentação Adium: artigos criados para auxiliar os usuários da aplicação, abrangendo uma ampla gama de temas, desde a configuração inicial, mensagens para o gerenciamento avançado da lista de contatos e personalização.

C. Help: criado para auxiliar a utilização do aplicativo;

D. Lista de Troubleshooting: fornece solução para os principais problemas que podem ser enfrentados com a utilização do aplicativo;

E. Lista das principais funcionalidades adicionadas à versão atual da aplicação;

F. Wiki: contendo as perguntas freqüentes e informações relevantes sobre a aplicação;

G. Doxygen: documentação do código e referências encontradas entre as classes.

14. Existem bugs marcados como fáceis de serem resolvidos?

Não. Não há esta tipificação no sistema de controle de *bugs*.

15. O código fonte contém listas de tarefas?

Não encontramos quaisquer listas de tarefas.

16. Existem comentários contendo TODO, FIXME, etc no código-fonte? Que tipo de necessidade estão colocadas nesses comentários?

Sim, existem comentários contendo TODO, abaixo um exemplo:

```
if (!resp) {
    NSLog(@"error decoding graph API response: %@", error);
    // TODO: indicate setup failed
    return;
}
```

Neste tipo de comentário estão indicadas algumas melhorias (e não funcionalidades básicas) que podem ser desenvolvidas no programa. Também existem alguns comentários contendo FIXME (3), segue um exemplo:

```
//FIXME: This class is not CodingStyle compliant.
@interface AIXMLElement : NSObject <NSCopying> {
        NSString *name;
        NSMutableArray *attributeNames;
        NSMutableArray *attributeValues;
        NSMutableArray *contents;
        BOOL selfCloses;
}
```

Não existem comentários do tipo OPTIMIZE, mas em alguns comentários é informado que foi usada algum tipo de variável, ou alguma determinada lógica para melhorar o desempenho do programa. Além dos citados, ainda são utilizados comentários com o marcador “???” para indicar uma dúvida, abaixo um exemplo:

```
case AINameFormat_ScreenName:
        //??? - How should this be handled for metaContacts?  What if there are no aliases set?
        formattedUID = inObject.formattedUID;
        longDisplayName = (formattedUID ? formattedUID : displayName);
        break;
```

Existe ainda outro tipo de comentário: “XXX”, onde os desenvolvedores indicam trechos de código que requerem análise e consequente refatoração, abaixo um exemplo:

```
//XXX - Re-evaluate this method and its presence in the core
- (BOOL)oneOrMoreConnectedOrConnectingAccounts
{
   for (AIAccount *account in self.accounts) {
      if (account.online || [account boolValueForProperty:@"isConnecting"] || [account
valueForProperty:@"waitingToReconnect"]) {
                        return YES;
      }
   }
        return NO;
}
```

17. Indique 5 tarefas que potencialmente serão realizadas até o final do curso. Estas tarefas podem ser relacionadas a resolução de bugs, implementação de funcionalidades, tradução, testes, revisão do código-fonte (i.e. arrumar código que não siga os padrões de estilo documentos do projeto), reportar bugs, etc. Outros tipos de tarefas podem ser realizados desde que aprovados pelos instrutores: fale com a gente.

Pela análise preliminar dos bugs disponíveis, achamos estes de complexidade muito alta para que possamos fazer alguma contribuição (todos os bugs mais simples estavam resolvidos ou já assinalados para alguém). A lista de funcionalidades a serem implementadas também apresenta apenas macro objetivos, ou melhor, não será possível contribuirmos nesta frente, pois demandaria muito tempo. Acreditamos, porém, que encontramos algumas estratégias para contribuição:

A. Pela análise do código fonte que realizamos até então, encontramos uma quantidade de código de teste pequena comparada com o tamanho do projeto, portanto entendemos que esta será uma grande oportunidade de contribuição. Imaginamos fazer uma análise de cobertura de testes e então definir partes do programa que precisam de testes automatizados, construiremos estes testes e submeteremos para a comunidade como contribuição.

B. Encontramos um conjunto de ferramentas implementado por um projeto OSS chamado “LLVM”. Entre outras coisas (http://clang-analyzer.llvm.org/), o LLVM tem uma ferramenta de análise estática de código fonte para a linguagem Objective C. Pretendemos usar esta ferramenta para apontar potenciais problemas, que reportaremos como *bugs* já associados a sua solução. Por uma análise inicial encontramos apenas 36 problemas com o Clang e é possível que representem apenas problemas menores e de muito baixo impacto.

C. Verificamos que nem todas as classes, métodos e propriedades (variáveis da classe acessíveis externamente) estão documentadas com o padrão HeaderDoc, que permite geração automática de documentação a partir dos

comentários em código fonte. Assim podemos complementar a documentação do código fonte usando esta sintaxe em classes de grande relevância.

IV. Caracterização Técnica

18. Quantos desenvolvedores participaram do projeto nos últimos 6 meses? E desde o início?

a) **Lista dos atuais "Lead Developers":** Evan Schoenberg, Zachary West;

b) **Project Manager:** Eric Richie;

c) **Lista dos Desenvolvedores:** Thijs Alkemade, Colin Barrett, Frank Dowsett, Adrian Godoroja, Peter Hosey, Matthew Needham, Patrick Steinhardt;

d) **Lista dos Colaboradores:** Paul Aurich, John Bailey, Adam Betts - Adiumy iconset, Moses Lei, David Munch, Nick Peshek, Jordan Schelew, Robert Vehse

e) **Suporte:** Andrew "proton" Wellington (libezv Bonjour support);

f) **User Interface:** Paul Wilde.

19. Qual a política de lançamento de versões?

Pelo fato da equipe ter escolhido trilhar pelo “admirável mundo novo” e o projeto não possuir uma política de lançamento de versões formalizada, acredita-se que será necessário mais convívio com as práticas adotadas para o projeto para responder a esta questão com maior propriedade de causa.

OBS: A equipe sugere que após essa pergunta, o questionário seja alterado para contemplar a seguinte pergunta:

- Existe alguma política de contribuição? Descreva-a.

A equipe identificou orientações no Fórum do Adium com a finalidade de nortear e sistematizar a troca de informações entre os colaboradores.

- Para solução de problemas:

A. Antes de postar um novo tópico ou responder pedidos relativos, características, erros, ou problemas com o Adium, por favor, procurar a documentação na Wiki Adium: WikiStart.

B. Na Wiki Adium também é possível encontrar "Perguntas Frequentes" que possam responder a sua pergunta (por exemplo, por que Adium atualmente não tem webcam ou voz / vídeo de apoio).

C. Confira as dicas de solução de problemas.

- Para reporte de *bugs*:

D. Leia as Regras do Fórum em primeiro lugar.

   As regras do fórum são fáceis de seguir e facilitam a vida dos outros usuários do fórum. Se você não as seguir em seu post, provavelmente, o mesmo será excluído.

E. Diga qual a versão do Adium você está usando, e em que sistema operacional.

   Você pode encontrar a versão em 'Sobre' diálogo no "Adium" menu.

F. O interessante é publicar uma receita, se possível. Como a proporção de usuários é muito maior que a de desenvolvedores, é provável que os desenvolvedores não encontrem em primeira mão o bug que você encontrou. Então, a sugestão dada é a de que: cada pessoa que descobrir como acionar o bug, passe a “receita”, passo a passo, para que os desenvolvedores consigar replicar e, assim, corrigi-los.

- Para reportar um *bug* / solicitação de funcionalidade:

G. Verifique se o seu pedido já foi criado:

   - Em "Hot questions" é a seção da página de Ajuda que lista os problemas mais comuns, relatados recentemente.
   - Pesquisar o bug tracker usando a pesquisa regular ou filtro “*Custom Query*”.

H. Se um bilhete com o seu problema já existe, basta acrescentar qualquer informação nova, julgada relevante para o seu caso em especial.

20. O projeto possui uma suíte de testes automatizados?

Sim, utiliza a funcionalidade de testes automatizados da IDE de desenvolvimento XCode.

21. Existem partes do código fonte que são problemáticas? Justifique.

Sim. Compilando o código fonte verificamos que há diretivas de compilação "#warning" com comentários associados indicando partes do programa que precisam de melhorias.

A análise estática com a ferramenta Clang, integrada à IDE XCode, mostrou apenas 26 alertas, o que é muito pouco para os mais de 70.000 linhas de código Objective-C. São alertas dos seguintes tipos:

A. Memory - Method returns an Objective-C object with a +0 retain count
B. Memory - Object over-autoreleased: object was sent -autorelease but the object has zero (locally visible) retain counts
C. Dead Store - Value stored to 'status' is never read
D. Memory - Potential leak of an object allocated on line 316
E. Memory - Object leaked: allocated object is not referenced later in this execution path and has a retain count of +1
F. Logic error - The left operand of '&' is a garbage value
G. Logic error - Looping back to the head of the loop
H. Logic error - The left operand of '&' is a garbage value

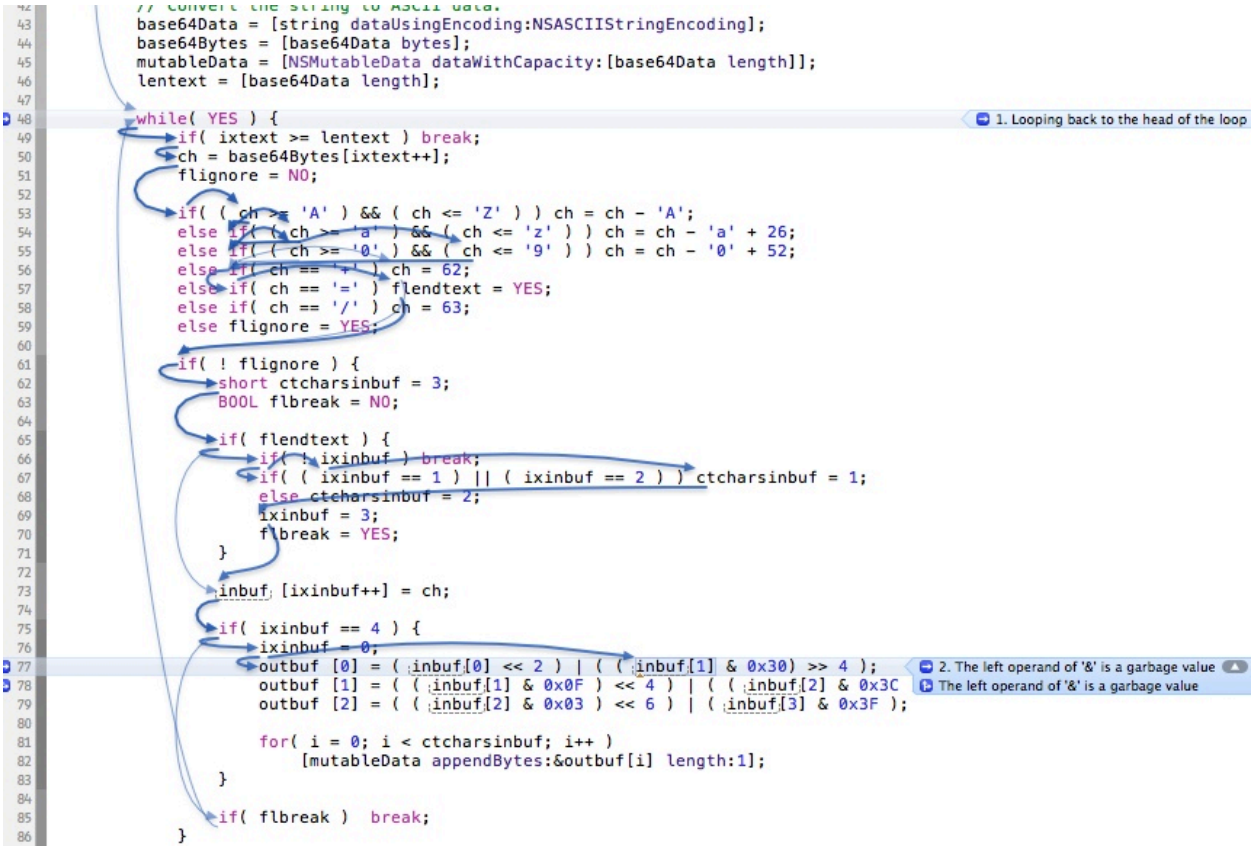

Figura 1 - Exemplo de respresentação gráfica da análise estática com o Clang

Alguns destes erros podem ser falsos positivos, pois referem-se a problemas de memória com contagem manual de referências (retain, release, autorelease), mas se o aplicativo for compilado com Garbage Collector, estes métodos não estarão sendo usados. Os demais problemas apontados de *Dead Store* e de *Logic Error* são problemas realmente potenciais, mas que possivelmente não estão gerando grandes impactos. Podemos realizar testes automatizados para comprovar os problemas de lógica e reportar Bugs a partir deles. Os Dead Store são pequenas otimizações de memória que podemos reportar, mas serão de baixa relevância se não estiverem em partes do programa que sejam de uso muito repetitivo.

22. O código contém comentários? Existem partes que necessitam de mais comentários?

Sim, há comentários e existem partes onde estes podem ser melhorados. Algumas definições de classes, de métodos e de atributos têm comentários. Há também comentários associados a alguns comandos. Para algumas definições de classe e métodos, foi utilizado a padrão de documentação compatível com o utilitário HeaderDoc, que permite a geração automática de documentação em HTML a partir dos comentários do código. Eis um exemplo para definição de uma classe: a diretiva @class e logo a seguir o @brief indicam a identificação e documentação de uma classe:

```
/*!
 * @class AIInterfaceController
 * @brief Interface controller
 *
 * Chat window related requests, such as opening and closing chats, are routed through the interface controller
 * to the appropriate component. The interface controller keeps track of the most recently active chat, handles chat
 * cycling (switching between chats), chat sorting, and so on.  The interface controller also handles switching to
 * an appropriate window or chat when the dock icon is clicked for a 'reopen' event.
 *
 * Contact list window requests, such as toggling window visibilty are routed to the contact list controller component.
 *
 * Error messages are routed through the interface controller.
 *
 * Tooltips, such as seen on hover in the contact list are generated and displayed here.  Tooltip display components and
 * plugins register with the interface controller to be queried for contact information when a tooltip is displayed.
 *
 * When displays in Adium flash, such as in the dock or the contact list for unviewed content, the interface controller
 * manages keeping the flashing synchronized.
 *
 * Finally, the interface controller manages many menu items, providing better menu item validation and target routing
 * than the responder chain alone would do.
```

```
 */
```

Neste caso vemos a sintaxe HeaderDoc para uma implementação de um método.

```
/*!
 * @brief Returns an array of available dock icon pack paths
 */
- (NSArray *) availableDockIconPacks
{ ...
```

Agora um método com parâmetros de entrada.

```
/*!
 * @brief Get the name and preview state for a dock icon pack
 *
 * @param outName Reference to an NSString, or NULL if this information is not needed
 * @param outIconState Reference to an AIIconState, or NULL if this information is not needed
 * @param folderPath The path to the dock icon pack
 */
- (void)getName:(NSString **)outName previewState:(AIIconState **)outIconState
forIconPackAtPath:(NSString *)folderPath
```

A seguir, dois exemplos de métodos (na mesma classe que os anteriores) sem HeaderDoc.

```
//Set an icon state from our currently loaded icon pack
- (void)setIconStateNamed:(NSString *)inName
//Remove an active icon state
- (void)removeIconStateNamed:(NSString *)inName
```

23. Padrões de projeto e padrões arquiteturais podem ser identificados a partir do código? Quais?

A linguagem Objective C tem comandos específicos para a utilização de alguns padrões de projetos. Muitos deles são padrões que viabilizam o padrão arquitetural MVC (Model - View - Controller). Seguem exemplificações de padrões de projeto encontrados no projeto Adium.

A. MVC - este padrão arquitetural é usado, havendo inúmeras classes do tipo controller, exemplo: AIAccountController, AIDockController. As classes do tipo view estão essencialmente usadas nos arquivos “NIB” e “XIB” de definição de interface, exemplo: AIModularPaneCategoryView, AIAccountSelectionView.

B. Delegation - usado para permitir retorno de informações de classes da camada model para controller ou mesmo entre controllers, sendo de um controller mais específico para um controller mais geral. Exemplo:

```
@interface AIDockController: NSObject <AIDockController, AIFlashObserver, AIChatObserver>
{
```

Aqui vemos que a classe “AIDockController” é filha (herança) de “NSObject” e implementa os protocolos “AIDockController”, “AIFlashObserver”, “AIChatObserver”, desta forma as instâncias desta classe poderão ser acionadas pelas instâncias das classes que definiram os respectivos protocolos referenciados.

No exemplo a seguir vemos o complemento da delegação, que é a instância de uma classe se definir como delegada para de outra instância, definidora do protocolo.

```
@implementation AIMenuController
- (id)init {
  if ((self = [super init])) {
          //Set up our contextual menu stuff
          contextualMenu = [[NSMenu alloc] init];
          [contextualMenu setDelegate:self];
          contextualMenuItemDict = [[NSMutableDictionary alloc] init];
          currentContextMenuObject = nil;
          textViewContextualMenu = [[NSMenu alloc] init];
          [textViewContextualMenu setDelegate:self];
  }
  return self;
}
```

C. Observing (por envio de mensagens) - existem recursos de envio de mensagens entre objetos para permitir comunicação entre camadas, mas neste caso temos uma comunicação assíncrona, ou melhor, ou melhor, o envio da mensagem não implica recebimento imediato pelos observadores (em KVO e Delegation o processo é síncrono). O sistema de mensagens também permite que uma mensagem alcance vários observadores, ou 1 para muitos (na delegação é uma relação de 1 para 1, em KVO podemos também ter relação de 1 para muitos). Um grande diferencial das mensagens em relação ao KVO é que este sistema permite acoplamento entre objetos de diferentes aplicações (no KVO é interno à aplicação):

Exemplo de definição de um observador:

```
[distributedNotificationCenter addObserver:self selector:@selector(applescriptRunnerIsReady:)
name:@"AdiumApplescriptRunner_IsReady" object:nil];
```

No exemplo acima o método que será executado é “applescriptRunnerIsReady:” e a mensagem que está sendo monitorada é "AdiumApplescriptRunner_IsReady" de qualquer objeto. É usado o sistema distribuído de mensagens, permitindo monitorar outras aplicações.

Exemplo de um objeto enviando mensagem para seus observadores:

```
[[NSDistributedNotificationCenter defaultCenter]
postNotificationName:@"AdiumApplescriptRunner_Quit" object:nil userInfo:nil
deliverImmediately:NO];
```

D. Observing (por monitoramento de valores de variáveis - KVO) - permite comunicação de 1 observado para vários observadores, é restrito ao contexto da aplicação e é síncrono. Exemplo:

```
// Track for item's selection changes
[self addObserver:self forKeyPath:@"selectionIndexes"
            options:(NSKeyValueObservingOptionNew)
```

```
                context:NULL];
```

E. Target - Action - permite a comunicação das classes da camada Controller com as classes da camada View. Target é usado para que as classes controller tenham acesso aos objetos da camada view. Action faz o oposto: permite que as classes da camada View acionem métodos dos objetos na camada Controller (eventos de interface chamando o controller para realizar tarefas). Este mecanismo é síncrono. No exemplo a seguir apresentamos alguns métodos que são acionados por objetos de interface em determinados eventos:

```
- (IBAction) closeMenu:(id)sender;
- (IBAction) closeChatMenu:(id)sender;
- (IBAction) closeAllChats:(id)sender;
```

Nos arquivos NIB e XIB (em formato XML) estão definidas as ligações de objetos (não classes, objetos específicos), em determinados eventos, com determinadas ações de um determinado controlador.

A seguir um exemplo de TARGET. O comando IBOutlet indica que esta variável poderá ser ligada a um objeto (mais uma vez não classe, mas sim objeto) de interface. Se ligado, esta informação estará contida nos arquivos NIB e XIB.

```
IBOutlet NSView      *fontPanelAccessoryView;
IBOutlet NSButton    *button_fontPanelSetAsDefault;
```

24. Como foi obtido o código fonte do projeto?

O projeto Adium utiliza um repositório com tecnologia Mercurial, portanto baixamos e instalamos um cliente da ferramenta de repositório para a plataforma Mac OS e executamos os comandos indicados no website do projeto para realizar a cópia do repositório:

prompt > hg clone http://hg.adium.im/adium

25. O que foi necessário instalar no seu sistema para fazer o projeto compilar/ rodar? A documentação disponibilizada pelo projeto foi suficiente?

Sim, a documentação foi suficiente. Como único requisito, precisamos ter instalada a IDE de desenvolvimento usada no projeto, chamada Apple XCode. A compilação funcionou sem maiores problemas, apenas com alguns *warnings*, sendo alguns, inclusive, propositais pelos desenvolvedores, com objetivo de alertar para alguns pontos de otimização.