ECLIPSE TOOLS - FERRAMENTA PARA AUXÍLIO À COMPOSIÇÃO DINÂMICA DE SOFTWARE

Lucas Albertins de Lima[1], Waldemar P. Ferreira Neto[2], Glauber Vinícius[3], Joseana Macêdo Fechine[4]

[1,2]Aluno do Curso de Ciência da Computação, integrante do PET-Computação, Depto. de Sistemas e Computação DSC/UFCG, Campina Grande, PB, {lucas,waldemar}@compor.net

[3]Aluno do Mestrado em Engenharia Elétrica, Coordenação de Pós-graduação em Engenharia Elétrica COPELE/UFCG, Campina Grande, PB, e-mail: glauber@compor.net

[4]Professora do Depto. de Sistemas e Computação DSC/UFCG, Tutora do PET-Computação, Campina Grande, PB, e-mail: joseana@dsc.ufcg.edu.br

RESUMO – Tendo em vista o desenvolvimento de uma das bases do Programa PET, a pesquisa, dois alunos do PET-Computação de Campina Grande iniciaram, junto com a *Nokia Embedeed*, um projeto de desenvolvimento de alguns módulos de uma ferramenta para manipular alguns artefatos para composição dinâmica de softwares. Este artigo aborda mais especificamente a construção de um *Builder* e *Properties* Views para um *plug-in* da plataforma Eclipse, com o intuito de realizar desenvolvimento baseado em componentes.

ABSTRACT – *Thinking about developing one of the bases of PET Program, the research, two students of PET-Computação from Campina Grande had initiated together with the Nokia Embedeed a development project of some modules to build a tool to manipulate some devices where the purpose is the dynamic composition of softwares. More specifically, this article approaches the construction of a Builder and Properties Views for a platform Eclipse plug-in with the intention of development based on components.*

(Palavras-chave: Componentes de Software, *Eclipse*, *plug-in* do *Eclipse*, Desenvolvimento baseado em componentes, *Properties View, Builder*, PET-Computação).

1. INTRODUÇÃO

Com o propósito de desenvolver a pesquisa do Grupo PET-Computação da UFCG (Universidade Federal de Campina Grande), dois alunos desse programa, iniciaram junto com a Nokia Embedeed, um projeto de desenvolvimento de alguns módulos de uma ferramenta para manipular alguns artefatos para composição dinâmica de softwares.

O desenvolvimento baseado em componentes tem sido apontado como promissor na construção de aplicações com maior capacidade de adaptação a mudanças nos seus requisitos. Porém, tal adaptação deve exigir esforço e tempo de implementação mínimos, para atender ao requisito de *time-to-market*[1] a que se submetem atualmente os sistemas de informação das empresas [HEINEMAN, 2001; SZYPERSKY, 1998].

Um componente é software auto-contido (*self-contained*), ou seja, somente define uma funcionalidade e não pode ser utilizado sozinho, e possui uma interface (ou

[1] Tempo de produção do software até que este seja colocado no mercado.

conjunto de interfaces) bem definida e documentada, que deve poder ser acessada em tempo de execução do software gerado pela composição.

O projeto COMPOR [COMPOR, 2005] visa à criação de todo um ambiente para o desenvolvimento de software baseados em componentes. O projeto é dividido em nove subprojetos de acordo com o contexto da pesquisa: aplicações, modelos de composição, ambientes da execução, estruturas, linguagens, métodos, repositórios, utilidades, e *workbenches*[2]. O foco da pesquisa ora apresentada são os modelos de composição, mais especificadamente o desenvolvimento de ferramentas para a modelagem da composição de softwares, através de ferramentas gráficas.

A ferramenta adotada foi o Eclipse, a partir da qual está sendo desenvolvido um *plug-in* para composição de softwares através dos componentes desenvolvidos pelo projeto COMPOR. Os módulos dessa ferramenta, que foram desenvolvidos e que estão sendo tratados neste artigo, são o *Builder* e os *Properties* do *plug-in* do Eclipse [Eclipse, 2005], como será descrito nas seções 1.3, 1.4 e 1.5.

1.1 Arquitetura de componentes COMPOR-CM

A arquitetura de componentes do modelo COMPOR-CM possui dois tipos de entidades: **contêineres** e **componentes funcionais**. Os componentes funcionais implementam as funcionalidades do sistema, disponibilizando-as em forma de serviços. Os componentes funcionais não são compostos por outros componentes, ou seja, não possuem "componentes-filhos". Os contêineres, por sua vez, não implementam funcionalidades, apenas gerenciam o acesso aos serviços dos seus "componentes-filhos". Ou seja, os contêineres servem como "portas de acesso" às funcionalidades dos componentes neles contidos. Na Figura 1 é apresentada uma arquitetura de sistema baseada em composição de módulos. Para tal arquitetura cada módulo é composto de sub-módulos que implementam as funcionalidades do sistema, mapeada em uma estrutura em que os nós-folhas representam as funcionalidades (componentes funcionais) e os outros nós (contêineres) representam "portas de acesso" às funcionalidades [Almeida, 2004].

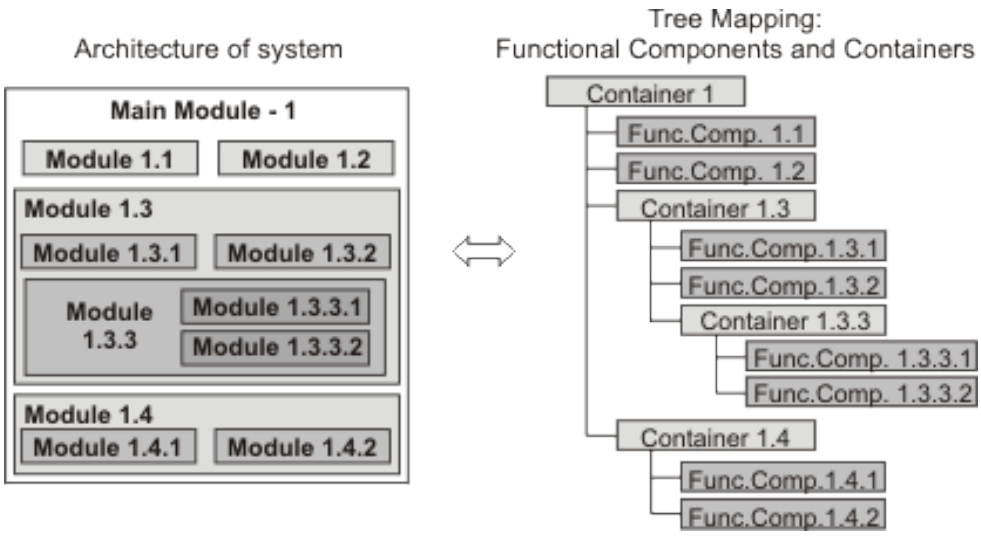

Figura 1 - Mapeamento em árvore da arquitetura de um sistema.

Com base neste mapeamento de arquitetura, o modelo COMPOR-CM promove a reutilização, tanto no nível de componentes funcionais como no nível de contêineres.

[2] Áreas de Trabalho onde poderão ser desenvolvidos softwares por composição através de uma interface gráfica.

Na Figura 2, por exemplo, apresenta-se a arquitetura clássica de software em três camadas na visão do modelo COMPOR-CM. Nesta figura, cada uma das camadas foi mapeada em módulos e suas funcionalidades em sub-módulos. O sistema (visto como um contêiner) é representado por três módulos (ou três contêineres), que por sua vez possuem componentes funcionais, os quais provêem os serviços do sistema. Visualizando um contêiner como uma caixa preta, a reutilização pode ocorrer no nível de sistema, para interoperabilidade com outros sistemas, por exemplo; no nível de módulos, caso ocorram modificações no tipo de interface do usuário ou estrutura de armazenamento, por exemplo; e no nível de componentes, caso alguma funcionalidade do sistema possa ser reutilizada em outros contextos [Almeida, 2004].

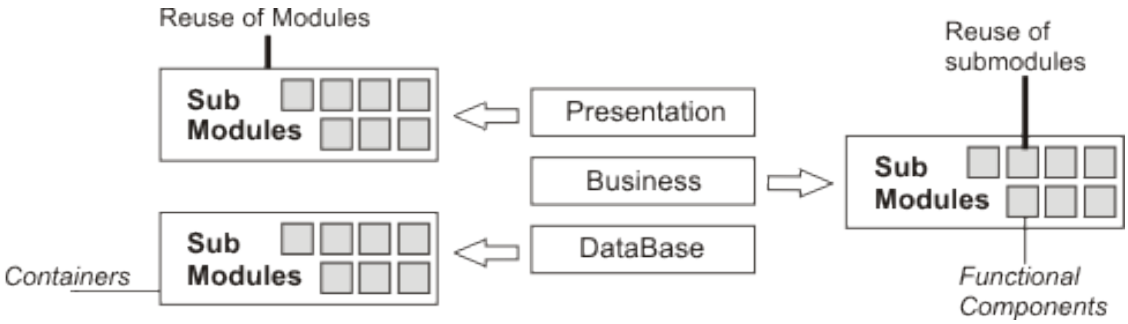

Figura 2 - Arquitetura clássica em três camadas mapeadas em módulos

1.2. Eclipse

O Eclipse é uma das ferramentas de desenvolvimento de software mais polulares atualmente para o desenvolvimento em plataforma Java, sendo considerada uma das ferramentas chave em se tratando de iniciativas *open-source* (código aberto). Como IDE[3], possui facilidades como visualização de todos os arquivos contidos no projeto de forma clara, ferramentas de gerenciamento de trabalho coletivo, compilação em tempo real, geração automática de código, dentre outras.

Em virtude do uso da tecnologia de *plug-in*s (que será detalhada com maior aprofundamento no próximo tópico), o Eclipse permite personalizar o ambiente de trabalho do desenvolvedor de acordo com o projeto que está sendo desenvolvido, seja ele, um simples projeto com páginas HTML estáticas, até aplicações com uso de EJBs (Enterprise Java Beans), *frameworks* diversos ou J2ME (*Java to MicroEdition*) [SUN, 2005]. Além disso, a tecnologia de *plug-in* possibilita a criação de seus próprios *plug-in*s.

Como *open source* há anos deixou de ser sinônimo de ferramentas sem recursos, com *bugs* e sem suporte algum, o Eclipse permite que se possa fazer em seu ambiente o mesmo que poderia ser feito em ferramentas pagas, como JBuilder [BORLAND, 2005].

Atualmente, o responsável pela continuidade do desenvolvimento da ferramenta é o Consórcio Eclipse.org [ECLIPSE, 2005], criado pela IBM, empresa responsável pelo desenvolvimento da ferramenta em sua fase inicial e depois disponibilizada como projeto *open-source*. Hoje o consórcio é formado por grandes empresas de tecnologia de software e desde 2004, tornou-se independente sem a influência direta da IBM [Eclipse, 2005].

O que faz desta ferramenta um diferencial é a flexibilidade proporcionada ao desenvolvedor. Ele sempre trabalha em um *workbench*, isto é, um ambiente que pode ser configurado conforme suas necessidades com uso de perspectivas (pode ser definida como o projeto é visto sob o olhar de uma plataforma ou ambiente), além de

[3] *Integrated Development Environment* – Plataforma de Desenvolvimento Integrada.

diversas *views* (visualizadores - ferramentas e recursos especiais para determinadas tarefas) e editores [Johan, 2003].

1.3 *Plug-in*

A idéia de *plug-in* está associada a uma extensão das funcionalidades da ferramenta, ou seja, "plugar" um pedaço de software ao que já existe, o qual irá possibilitar ao desenvolvedor realizar atividades além das que já eram possíveis. Com isso, pode-se adicionar recursos a um ambiente, criar novos ambientes de programação para outras linguagens ou para um propósito específico [COMPOR, 2005].

1.4 Builder

Um *Builder* é um dispositivo que processa um conjunto de recursos para realizar determinadas tarefas específicas, que serão realizadas de acordo com a implementação feita para cada recurso, em que recursos podem ser arquivos, pastas, projetos relacionados a um *workspace*[4]. Por exemplo, um *Java Builder* recompila mudanças feitas em arquivos Java e produz arquivos *.class*. A idéia é ter *Builders* incrementais que verifiquem mudanças ocorridas durante o projeto e faça um processamento necessário de acordo com as regras de cada ferramenta.

Builders são configurados antes dos projetos, são associados a estes e rodam automaticamente quando houver mudanças nos seus recursos. São freqüentemente usados para aplicar transformações em recursos para produzir artefatos de outros tipos.

As ações do *Builder* são invocadas através da chamada do método *build()*, a partir dessa ação, verifica-se o tipo de *build* que se deseja acionar.

Existem dois tipos de *builds* [Arthorne, 2005]:

- **Full Build** – este tipo faz um *build* do zero. Ele trata cada recurso do projeto como se nunca tivesse sido visto pelo *Builder* e realiza o processamento necessário em todos.

- **Incremental Build** – usa sempre o último estado do build, mantido internamente pelo *Builder*, para realizar um *build* otimizado no qual só é verificado o que foi mudado desde a última ação do *Builder*. Esta última mudança é chamada de *delta* e é em torno deste delta que vai acontecer o processamento do *Builder*.

Existe ainda um outro tipo, que pode não ser considerado um *build* propriamente dito, pois sua ação é apenas descartar todo tipo de artefato gerado pelo *Builder*, ele é chamado de **Clean Build**.

[4] Área de Trabalho, onde estão localizadas as principais funcionalidades para a execução e uma tarefa nessa ferramenta.

1.5 *Properties View*

O *workbench* do eclipse fornece muitas visões, mas uma delas é bastante versátil, a *Properties View* ou visualizador de propriedades. Esta fornece uma maneira de exibir as propriedades de um item do *workbench* selecionado. Um *Properties View* pode exibir somente informações de leitura (não editávies), tal como as propriedades de um arquivo de recursos ou pode exibir as informações que podem ser editadas, como as propriedades de um ponto da extensão em arquivo xml. Um *Properties View* pode mostrar propriedades para vários itens. Um item pode ser um recurso do diretório, um recurso de arquivo, ou um elemento dentro de um recurso (tal como uma extensão em um arquivo de *plug-in* .xml) [Johan, 2003].

2. METODOLOGIA

A metodologia utilizada no projeto foi bastante simples. Inicialmente, o gerente da equipe indicou material para leitura, principalmente sobre a plataforma Eclipse e sobre desenvolvimento de *plug-ins* para a mesma [GAMMA, 2004; ARTHORNE, 2004 e SHAVOR, 2003]. Tomando como base esse material, foram elaboradas duas palestras sobre as funcionalidades a serem desenvolvidas: o *Build*, com o título "*Build in Eclipse*"; e o *Properties View*, com título "Tomando controle do *Properties View*". Esse conjunto de palestras foi oferecido para o gerente e toda a equipe do projeto, e sendo realizada, ao término de cada uma, uma avaliação feita pela equipe e pelo gerente.

Após essa fase de sedimentação do conhecimento, deu-se inicio à fase de implementação das funcionalidades do *plug-in*. Todas as tarefas a serem cumpridas nessa etapa foram bem definidas pelo gerente, usando a ferramenta *xPlanner*[5]. O conteúdo dessas tarefas não pode ser publicado por uma política interna do projeto. O cumprimento de cada uma das tarefas é monitorado através da mesma ferramenta, *xPlanner*, como também em reuniões semanais com os gerentes para esclarecimentos de eventuais dúvidas e cobranças.

Para essa fase de implementação estão sendo dedicadas 6 horas semanais de cada participante do projeto, podendo, cada um, definir da forma que achar mais interessante o seu horário, exceto o horário da reunião semanal.

3. APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

O desenvolvimento baseado em componentes é o foco do projeto, sendo para isso necessário uma interface gráfica de acordo com [Heineman, 2001].

Apoiando-se no conceito de *plug-in*, deu-se início ao projeto para a construção de um *plug-in* que dê apoio à composição dinâmica de software baseada na especificação do modelo de componentes abordada anteriormente, o projeto CCT (*Component Composition Tools*), conforme Figura 3.

Este projeto tem por finalidade a construção de um ambiente de programação construído sobre o núcleo da plataforma Eclipse, em que o desenvolvedor irá poder adicionar seus componentes e fazê-los disponíveis para o reuso (*component pallete/manager*), gerenciar e configurar os componentes da aplicação (*componenttree/inspector*), realizar testes de integração (*component test*) e descrever novos componentes (*component description wizard*). Além disso, um editor de componentes e um editor de aspectos estão sendo elaborados sobre as ferramentas

[5] XPlanner é uma ferramenta de planejamento e acompanhamento de projetos. Com a qual os gerentes podem colocar as tarefas e a carga horária planejada ou verificar os horários dos desenvolvedores e os desenvolvedores podem verificar as tarefas e colocar as horas trabalhadas [XPlanner, 2005].

de desenvolvimento na linguagem *AspectJ* para facilitar o desenvolvimento de atividades sobre o componente.

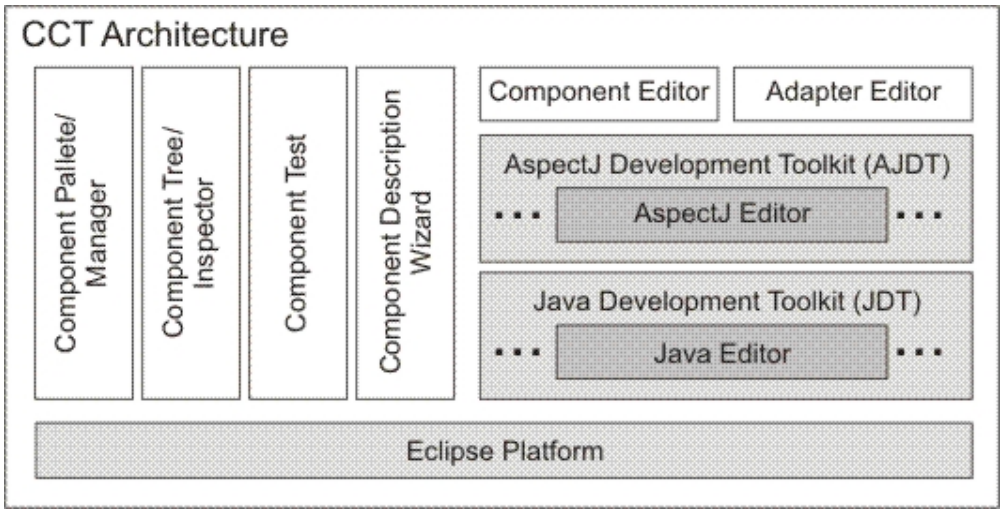

Figura 3 - Estrutura do *plug-in*.

A interface gráfica do CCT é implementada com um conjunto de *views* do Eclipse. Também foi criada uma perspectiva CCT para prover ao desenvolvedor uma área de trabalho em componentes customizável. Na Figura 4 é apresentada a principal tela da perspectiva CCT.

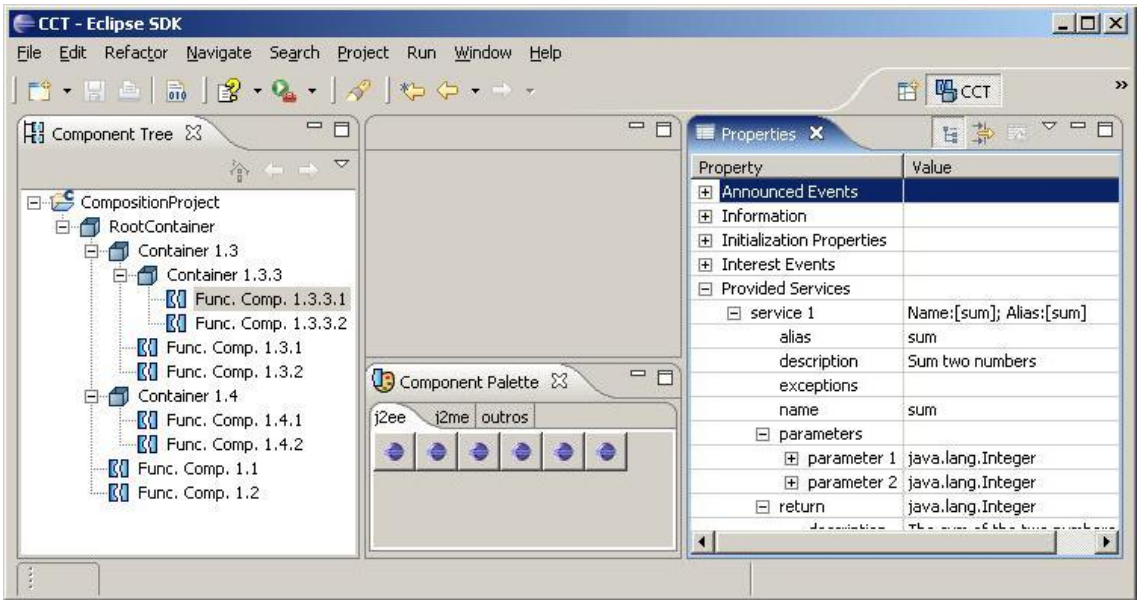

Figura 4 - Interface gráfica do CCT no Eclipse.

Dentre todas as funcionalidades do *plug-in*, que está em fase de desenvolvimento, as que foram desenvolvidas pelos alunos do programa PET, e que, portanto, serão retratadas neste artigo, foram o *Builder* e os *Properties*, descritos a seguir.

3.1 Builder

Com relação ao *Builder*, inicialmente foi desenvolvido um trabalho de estudo, para os desenvolvedores se familiarizem com os conceitos relacionados a *Builder Projects*. A partir deste estudo foi feito um seminário sobre o que foi pesquisado, realizado no laboratório do projeto *Embedded* na UFCG e, em seguida, foi discutida a idéia de como se implementar o CCT *Builder*.

Baseado no conceito de *Builder* foi planejado o desenvolvimento, para o *plug-in* do projeto CCT, um *Builder* chamado *CCT Builder*, cuja ação é verificar os recursos de um *workspace* CCT, verificar a existência de erros, validar os recursos que se referem a componentes e copiar para uma pasta de saída os recursos verificados pelo *Builder*, mantendo a hierarquia de classes do projeto.

O *Builder* deverá verificar basicamente três tipos de arquivos, com as seguintes extensões: *cctproject*, que é o arquivo de configuração de um projeto CCT, fica na raiz do projeto, *.abc* que são arquivos que referenciam componentes existentes e o *.def* que são arquivos de registro de *containers*[6].

Após isto, deu-se início ao desenvolvimento. Primeiro foi dada ênfase na ação de *Full Build*, em que o *Builder* "varre" os recursos de um projeto e faz o processamento necessário. Primeiro, o *Builder* projetado retira do arquivo *.cctproject* a localização da pasta de saída, em seguida o copia para está pasta de saída. Depois é verificada toda a hierarquia de recursos e copiada para esta pasta de saída. No caso de arquivos *.abc* é verificado se o componente referenciado por ele existe na paleta de componentes, caso não exista é adicionada uma marca de problema (*Problem Marker*) ao arquivo e o mesmo não é copiado para a pasta de saída. Caso haja algum erro de sintaxe, o mesmo procedimento é realizado. Em relação aos arquivos *.def* são verificados apenas os erros de sintaxe.

Logo após o *Full Build,* foi implementado o *Incremental Build*. Da mesma forma que o anterior, só que relacionado aos recursos do *delta,* que é a última mudança no estado do projeto.

Por fim, o *Clean Build* foi feito de acordo com o que deve ser o seu comportamento. Ele destrói todos os artefatos colocados na pasta de saída e remove todas as marcas de problema colocadas nos arquivos do projeto.

Atualmente está sendo desenvolvido um *Help* para o *Builder* e estão sendo realizados os testes automáticos para o mesmo.

3.2 Properties View

No caso do *Properties View* desenvolvido, os itens são os *containers* e os componentes funcionais da Arquitetura de componentes COMPOR-CM. No *workbench*, quando um desses itens for selecionado, o *Properties View* deve exibir as propriedades relevantes desses componentes, da forma como está descrito no Quadro 1. Os nomes em negrito são os tipos de componentes, ou seja, os tipos de item do *workbench*; os nomes que estão logo abaixo dos componentes são as suas propriedades, as propriedades que forem precedidas por **+** significa que pode haver mais de uma delas; e as que estiverem identadas significa que são sub-propriedades da anterior. Cada componente forma assim uma árvore de propriedades.

Quadro 1 – Descrição das propriedades dos componentes.

[6] *Containers* é uma estrutura que encapsula outras subestruturas mistas de um componente da COMPOR [COMPOR, 2005].

+Announced Events	**+Interest Events**
+ event 1	+ event 1
Description	Alias
Name	Description
Alias	Name
+ parameters	+ parameters
+ parameter 1	+ parameter 1
Description	Description
Type	Type
+Information	**+Provided Services**
Class name	+ service 1
Description	Alias
Host	Description
Identifier	+ exceptions
Name	+ exception 1
+Initialization Properties	Description
+ property 1	Type
Description	Name
Name	+ parameters
Value	+ parameter 1

+Required Services	Description
+ service 1	Type
Description	+ return
Alias	Description
+ exceptions	Type
+ exception 1	
Description	
Type	
Name	
+ parameters	
+ parameter 1	
Description	
Type	
+ return	
Description	
Type	

Todas essas propriedades são provenientes do arquivo do tipo *.abc* que descreve o componente. A maioria delas não é editável, as únicas que têm essa característica são os *alias* de todos os componentes, menos o **Information**, pois não possui essa propriedade; outra propriedade editável é o *host* de **Information.**

4. CONSIDERAÇÕES FINAIS

Com relação ao *Builder*, todas as funcionalidades intrínsecas a esse conceito com relação ao projeto já foram implementadas [ARTHORNE, 2005], agora está sendo desenvolvida uma série de testes para verificar a robustez da implementação e identificar eventuais *bugs*, além de está sendo elaborado também a documentação dessa funcionalidade para o esclarecimento de possíveis dúvidas de usuários do projeto ou para esclarecimentos de futuras modificações da ferramenta.

No que se refere ao *Properties View*, o modo de exibição e edição dos atributos dos *containers* e componentes de software já foi implementado. Falta ainda desenvolver um módulo para salvar as alterações na interface gráfica em arquivo, e recuperar essas alterações ao iniciar o *plug-in*. Além de, como o *Builder*, desenvolver uma série de testes para as funcionalidades implementadas e elaborar a documentação dos módulos.

Diante do exposto, pode-se constatar que a funcionalidades implementadas estão de acordo com as especificações fornecidas. Dessa forma, a equipe está cumprindo de forma adequada a sua atividade de pesquisa no âmbito do grupo PET.

5. REFERÊNCIAS BIBLIOGRÁFICAS

1. [ALMEIDA, 2004] H. O. de Almeida, A. Perkusich, E. de Barros Costa, and R. de Barros Paes. Composição Dinâmica de Componentes para Aplicações com Mudanças Freqüentes de Requisitos. In *Brazillian Component Workshop*, 2004.

2. [ARTHORNE, 2004] Arthorne, John; Laffa, Chris. *Official eclipse 3.0 FAQs*. Addison-Wesley, 2004.

3. [ARTHORNE, 2005] Arthorne, John. *Project Builders and Natures*, Eclipse Technical Articles, Novembro de 2004.

4. [BORLAND, 2005] Borland Software Corporation. Borland JBuilder. http://www.borland.com/us/products/jbuilder/index.html. Último acesso em 07 de novembro de 2005.

5. [COMPOR, 2005] COMPOR - Software Composition. http://wiki.compor.net/ index.php/ Home - Último acesso em 07 de novembro de 2005.

6. [ECLIPSE, 2005] Eclipse Foundation. http://www.eclipse.org. Último acesso em 07 de novembro de 2005.

7. [ECLIPSE HELP, 2005] Eclipse Foundation, Plataform Plug-in Developer Guide, *Incremental project builders.* http://help.eclipse.org/help31/ index.jsp?topic=/org.eclipse.platform.doc.isv/guide/resAdv_builders.htm. Último acesso em 07 de novembro de 2005.

8. [GAMMA, 2004] Gamma, Erich; Beck, Kent. *Component Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, 2004.

9. [HEINEMAN, 2001] G. T. Heineman and W. T. Councill. *Component Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, 2001.

10. [JOHAN, 2003] Johan, Dicky. *Take Control of Your Properties*, Eclipse Technical Articles, 2003.

11. [SHAVOR, 2003] Shavor, Sherry; D'Anjou, Jim; Fairbrother, Scott; Kehn, Dan; Kellerman, John; McCarthy, Pat. *The Java™ Developer's Guide to Eclipse*. Addison-Wesley, 2003.

12. [SUN, 2005] S. Microsystems. Java Technology. http://java.sun.com Último acesso em 07 de novembro de 2005.

13. [SZYPERSKY, 1998] C. Szypersky. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

14. [XPlanner, 2005] XPlanner Organization. http://www.xplanner.org/. Último acesso em 07 de novembro de 2005.