Desenvolvimento Visual de Arquiteturas Distribuídas

Robson G. F. Feitosa Renato L. C. Lima Cidcley T. de Souza
Centro Federal de Educação Tecnológica do Ceará, Gerência de Telemática
NASH – Núcleo Avançado de Engenharia de Software Distribuído
e Sistemas Hipermídia
Fortaleza-CE, Brazil, 60040-531
cidcley@cefetce.br, robsonf@gmail.com, macaroots@gmail.com

Resumo

As vantagens fornecidas pela utilização dos conceitos de arquitetura de software e estilos arquiteturais distribuídos, muitas vezes, não são de fato aproveitadas pela carência de ferramentas que suportem esses conceitos. Além disso, as atuais linguagens de descrição de arquiteturas não fornecem as características requisitadas pelas principais infra-estruturas de middleware onde essas aplicações de fato serão executadas. Nesse sentido, apresentamos neste trabalho, um ambiente de desenvolvimento de aplicações distribuídas que permite a elaboração de complexos projetos arquiteturais e realiza a geração automática de códigos diretamente em infra-estruturas de middleware como CORBA.

Palavras chaves: Arquitetura de Software, Sistemas Distribuídos, CORBA, Desenvolvimento Visual.

Abstract

The advantages obtained by applying the concepts of software architecture and architectural styles, very often, are not realizes due to the lack of tools to support such concepts. In addition, the current architectural description languages do not provide the characteristics required by the leading middleware infrastructures where these applications are in fact executed. Therefore, we present in this paper a developing environment for distributed applications which allows for the delivery of complex architectural projects and performs automatic code generation directly to middleware infrastructures such as CORBA.

Keywords: Software Architecture, Distributed Systems, CORBS, Visual Development.

1. INTRODUÇÃO

Arquitetura de software [14, 2] tem se mostrado uma disciplina importante por desempenhar um papel essencial no processo de desenvolvimento do software, separando os elementos de computação (componentes) dos mecanismos de comunicação (conectores). Esta separação permite e facilita a promoção do reuso em níveis abstratos, que é um dos principais benefícios fornecidos pelo desenvolvimento arquitetural. Entretanto, para que esses benefícios da arquitetura de software sejam realmente alcançados, ela deve ser tratada explicitamente servindo como base para análise, projeto e implementação. Nesse contexto, destacamos as ADLs (*Architecture Description Languages*) para a realização dessa tarefa.

Linguagens de Descrição de Arquitetura (ADLs) são notações semi-formais usadas na representação e análise de arquiteturas. Essas linguagens têm ênfase na estrutura de alto nível da aplicação em vez de se deter a detalhes de implementação [13]. ADLs têm se tornado, recentemente, uma área de pesquisa intensa na comunidade de arquitetura de software. Diversas ADLs têm sido propostas para modelagem de arquiteturas, tanto para domínio específico como para propósito geral. Podemos citar, por exemplo, ACME, Aesop, C2, Darwin, MetaH, Rapide, SADL, UniCon, Wright, CL, entre outras.

Sob o ponto de vista das aplicações distribuídas, o processo de desenvolvimento se dá, basicamente, pela adoção de infra-estruturas de *middleware* que forneçam serviços que permitam o desenvolvimento de aplicações complexas sem a preocupação com os fatores de distribuição em si. Arquiteturas como, RMI [11], CORBA [10], entre outras, têm fornecido mecanismos realmente poderosos para a criação de aplicações distribuídas complexas.

Contudo, em [5], podemos encontrar um conjunto de exemplos que mostram que as ADLs atuais não fornecem mecanismos que permitam a utilização de conceitos arquiteturais no desenvolvimento de aplicações distribuídas. Também em [3], foi realizado um estudo aprofundado sobre a indução de estilos na utilização de infra-estruturas de *middleware*. Nesse estudo, foi concluído que as ADLs atuais carecem de características que permitam a implementação de suas especificações e que as infra-estruturas de *middleware* ainda não são compatíveis com as especificações de arquiteturas desenvolvidas por ADLs.

Para contornar esse problema, apresentamos em [15] um *framework*, denominado DraX (*DistRibuted Architecture based on XML*). DraX é formado por um conjunto de ferramentas baseadas em tecnologias de larga aceitação no mercado e na academia para a especificação, verificação, análise e implementação de arquiteturas de software e de estilos arquiteturais distribuídos. Mostraremos que essas ferramentas podem realmente capturar os conceitos clássicos de arquiteturas de software e estilos arquiteturais, e podem fornecer meios para que as especificações criadas possam gerar implementações baseadas em infra-estruturas de *middleware*, onde todo o potencial dessas plataformas pode ser utilizado.

Muito embora o *framework* DraX forneça mecanismos para a implementação de arquiteturas de softwares distribuídos sobre infra-estruturas de middleware, esse *framework* ainda não permite uma boa produtividade em termos de geração de implementações. Esse problema se dá por conta da necessidade da manipulação, por parte do desenvolvedor, de um grande número de linguagens e scripts utilizados para a produção de aplicações. Nesse sentido, estamos propondo neste trabalho a utilização de uma ferramenta visual para automatizar o desenvolvimento de aplicações com DraX, de forma a elevar a produtividade na utilização desse *framework*. Essa ferramenta foi denominada GUIDraX.

Sugerimos, neste trabalho, que uma representação visual específica para representar aspectos arquiteturais, também específicos, pode facilitar a comunicação entre os *stakeholders* envolvidos no desenvolvimento de aplicações distribuídas complexas. Assim, em GUIDraX, fornecemos uma gramática visual para representar os principais aspectos relativos ao processo de descrição de arquiteturas e estilos arquiteturais distribuídos.

Além desta seção, apresentamos, na seção 2, um conjunto de trabalhos relacionados. Na seção 3, apresentamos um resumo sobre o *framework* DraX. Na seção 4, apresentamos a gramática visual de GUIDraX. Na seção 5, apresentamos a ferramenta GUIDraX. Na seção 6, mostramos a utilização de GUIDraX através de um estudo de caso. Finalizamos, na seção 7, com algumas conclusões.

2. TRABALHOS RELACIONADOS

A produção de ferramentas de suporte a descrição de arquiteturas de software ainda é uma área muito carente em termos de resultados práticos. De fato, as contribuições mais relevantes nessa área dizem respeito à produção de novas ADLs e estilos para suporte a construção de aplicações. Uma das poucas exceções é a ferramenta AcmeStudio [7], que foi desenvolvida pela Carnegie Mellon University e fornece uma interface gráfica que permite projetar arquiteturas de software. Essa ferramenta é baseada na terminologia da ADL ACME [6] que não dá suporte à geração de código para aplicações distribuídas e, portanto, AcmeStudio também não suporta essa característica.

Além de AcmeStudio, uma outra contribuição da Carnegie Mellon University é o AESOP [8] (*Software Architecture Design Environment Generator*). AESOP fornece um *toolkit* para a geração de arquiteturas de domínio específico. Contudo, da mesma forma que AcmeStudio, AESOP não fornece suporte a arquiteturas distribuídas nem a geração de código.

3. O *FRAMEWORK* DRAX

O *framework* DraX (*DistRibuted Architecture based on XML*) é formado por um conjunto de linguagens, esquemas e *scritps* que podem ser utilizados em conjunto para permitir a descrição, validação, análise e implementação de arquiteturas de software e de estilos arquiteturais distribuídos sobre infra-estruturas de *middleware*.

3.1 Requisitos para o *framework* DraX

Além das óbvias características necessárias para a elaboração de projetos arquiteturais disponíveis em todas as ADLs (para uma descrição detalhada sobre essas características consultar [13]), resolvemos incluir em DraX um conjunto de características que pudessem tanto facilitar a utilização dos conceitos de arquitetura de software e de estilos

arquiteturais por profissionais da indústria de software que não estão familiarizados com as notações das ADLs, como também fornecer mecanismos para que as especificações possam ser facilmente implementadas em alguma arquitetura de middleware sem que seja necessário o domínio dessa arquitetura pelos desenvolvedores. Dessa forma, diversos requisitos devem ser atendidos pelas linguagens, scripts e esquemas de DraX para conseguir atingir os objetivos que traçamos. Dentre esses requisitos podemos destacar os seguintes:

1. **Descrição da Arquitetura do Software Distribuído de forma Distribuída**: Em DraX, propomos a idéia de construir essa descrição arquitetural também de forma distribuída, no sentido de que os componentes que formam a arquitetura possam ser criados de forma distribuída sendo referenciados no momento da definição da arquitetura. A ADL que propomos em DraX, que denominamos ArchML, possui a característica de permitir a descrição separada, e, possivelmente, distribuída, dos componentes que formam a arquitetura.

2. **Descrição de Estilos Arquiteturais**: Convencionalmente, a utilização de estilos arquiteturais nas ADLs se dá pela utilização de descrições arquiteturais parametrizadas [4], como em Darwin [12] e Wright [1]. Diferente dessa abordagem, em DraX, para conseguirmos uma melhor separação de interesses na especificação de estilos e arquiteturas, desenvolvemos uma linguagem específica para a descrição de estilos arquiteturais, que denominamos Xtyle. Com Xtyle podemos construir novas especificações de estilos, refinar especificações existentes e até mesmo criar especificações baseadas em informações sintáticas de outros estilos, herdando algumas de suas características.

3. **Utilizar um ORB para representar Conectores**: A intenção de DraX é produzir projetos de arquiteturas distribuídas passíveis de implementação sobre algum *middleware*. Dessa forma, resolvemos não utilizar definições explícitas de conectores nas linguagens de DraX, visto que implicitamente as especificações serão implementadas tendo um ORB como representação desses conectores. Desse modo, podemos dizer que ArchML é uma ADL que utiliza a idéia de configuração *in-line*. Como a intenção de DraX é produzir software distribuído, e para isso devemos lançar mão de algum *middleware*, a adoção de um ORB para representar os conectores é uma solução adequada, mesmo restringindo as especificações às funcionalidades do ORB subjacente. Contudo, se por exemplo, utilizarmos um ORB CORBA, ainda assim temos uma gama razoável de estilos de comunicação, que podem ser síncrono, assíncrono, baseado em mensagens, baseado em eventos, entre outros. Para fortalecer ainda mais esse conceito, são apresentados em [5] argumentos que explicam que para a maioria das aplicações distribuídas baseadas em *middleware* é completamente desnecessário a descrição explícita desses conectores.

3.2 Metodologia de Desenvolvimento

De forma a contemplar todos os requisitos apresentados anteriormente, o *framework* DraX é formado por um conjunto de linguagens e ferramentas que dão suporte a todo o processo de desenvolvimento baseado em arquitetura, que vai desde a especificação da arquitetura e do estilo arquitetural, passando pela fase de verificação de especificações, até a chegar à geração de código. Desse modo, como o ferramental de DraX é relativamente extenso, resolvemos criar uma metodologia que possa guiar os passos de desenvolvimento de arquiteturas com o *framework*.

Podemos visualizar duas etapas de desenvolvimento distintas e paralelas que interagem entre si. Na primeira etapa a arquitetura da aplicação é descrita; na segunda etapa o estilo arquitetural é descrito, caso esse ainda não tenha sido especificado. O resultado do desenvolvimento de um estilo é sua especificação em Xtyle devidamente validada. Essa especificação poderá ser referenciada na etapa de descrição de uma arquitetura ArchML. A tarefa de validação dessa arquitetura, entre outras coisas, verificará a aderência da mesma ao estilo referenciado.

É bom observar que a etapa de desenvolvimento de estilos não é comumente usada, pois DraX possui uma base de estilos pré-definidos que podem ser referenciados para criar arquiteturas, entretanto, é um diferencial de DraX com relação a outras ADLs e/ou ambientes de descrição de arquiteturas, o fato de o desenvolvedor contar com uma linguagem específica para definir seus próprios estilos arquiteturais. Além disso, esses novos estilos tanto podem ser definidos a partir de refinamentos dos estilos pré-existentes em DraX, como podem ser totalmente definidos.

De qualquer modo, a metodologia geral que propomos para a utilização de DraX para especificação de arquiteturas distribuídas é composta pelas seguintes etapas:

1. **Especificação**: Nessa fase, são especificadas as arquiteturas e os estilos arquiteturais utilizando as linguagens ArchML e Xtyle respectivamente;

2. **Validação**: A verificação sintática, estrutural e comportamental das arquiteturas e estilos é realizada nessa fase. Sendo que são utilizados os esquemas e *scripts* de validação e verificação de DraX;

3. **Geração de Código**: Os *templates* de implementação são gerados a partir das especificações arquiteturais previamente validadas e das informações dos estilos que essas arquiteturas utilizam.

É importante ressaltar nesse ponto que não estamos apresentando um processo de desenvolvimento de software distribuído baseado em arquitetura. Mesmo sabendo que apresentamos uma metodologia de aplicação das ferramentas de DraX para a produção de software distribuído baseado em arquitetura, não nos preocupamos em formalizar os passos e interações relativas a essas ferramentas. Dessa forma, por exemplo, aspectos como refinamento de especificações arquiteturais [6] não são considerados nesse trabalho, mesmo sabendo que essa etapa pode ser realizada por ocasião da construção das especificações ou até mesmo diretamente nos códigos gerados a partir das ferramentas de DraX. Nas seções seguintes, mostraremos as fases de especificação e validação sintática de arquiteturas e estilos arquiteturais. Por carência de espaço, não apresentamos aqui as demais fases da elaboração de arquiteturas distribuídas.

4 A GRAMÁTICA VISUAL DE GUIDRAX

Para facilitar a utilização de alguns conceitos da arquitetura de software de forma visual, como aparência de um componente e seus atributos (interfaces, portas, conexões) e algumas regras de comunicação, como modo de sincronismo e tipo de conexão, foi definida uma gramática visual que compõe o ambiente de desenvolvimento visual da Ferramenta GUIDraX.

4.1. Representação Visual de Componentes

Componentes são as estruturas utilizadas para agrupar as informações de uma unidade do sistema. Segundo a linguagem ArchML, usada no *framework* DraX, um componente é composto dos seguintes elementos:

```
+ document (?)
+ propertSet (?)
- interfaces (+)
       role
        + port (+)
- behavior
```

Vale ressaltar que alguns desses elementos, como `document` e `properSet`, são atributos de caráter opcional, e conseqüentemente sem notação visual, apenas textual.

Os elementos associados com “+”, “-” ou “” antes de sua declaração são representações de um elemento que tem filhos não mostrados; um elemento que exibe seus filhos logo abaixo dele; e um nó (elemento sem filhos), respectivamente. Já os símbolos “(?)”, “(+)” e “(*)” associados logo após a declaração dos elementos significam: o elemento aparece zero ou uma vez; o elemento aparece uma ou mais vezes; e o elemento aparece zero ou mais vezes, respectivamente.

Assim, a notação visual definida para representar um componente é um retângulo com as bordas arredondadas. O nome do componente é apresentado a baixo do retângulo (Figura 1).

Nome do Componente

Figura 1. Representação Visual de um Componente

4.2 Representação Visual das Interfaces

Interfaces são encarregadas de descrever as portas que representam os serviços prestados ou utilizados pelos componentes. Sua notação visual é um retângulo com bordas ortogonais, com o nome do papel (role) declarada ao lado do retângulo (Figura 2). Dentro de uma interface pode-se definir uma ou mais portas.

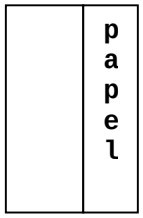


Figura 2. Representação Visual de uma Interface

4.3 Representação Visual das Portas de Comunicação

Portas são os pontos onde ocorrem trocas de informações entre os componentes. Sua representação visual pode ser analisada na Figura 3:

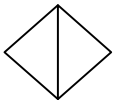
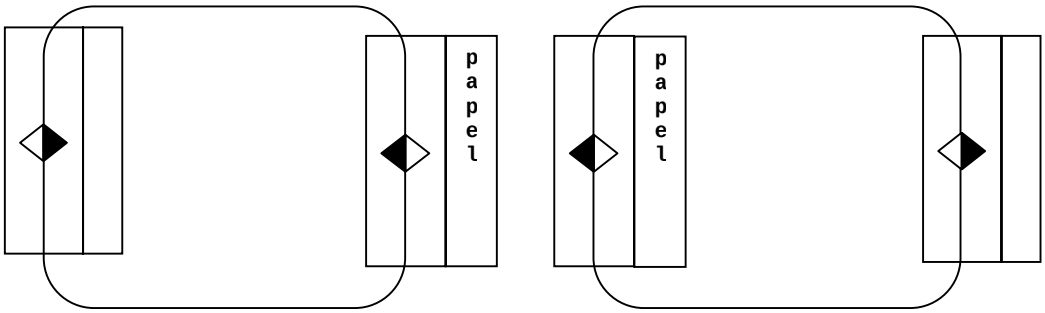


Figura 3 - Representação Visual de uma Porta

Porém, uma porta pode ser classificada como: Porta de Entrada “in” (representada na Figura 4a) ou Porta de Saída “out” (representada na Figura 4b).



(a) (b)

Figura 4. Representação Visual de Portas de Entrada (a) e portas de Saída (b)

Ou seja, uma porta com a parte preenchida do losango voltada para dentro do componente representa uma Porta de Entrada. Já uma porta com a parte preenchida do losango para fora do componente representa uma Porta de Saída.

Para representar o modo de comunicação das portas foi definida a seguinte notação visual: síncrono (*sync*, *ver Figura 5a*) ou assíncrono (*async*, *ver Figura 5b*).

(a) (b)

Figura 5. Representação de Portas Síncronas (a) e portas Assíncronas (b)

4.4. Representação Visual de Conexões
A linguagem Xtyle fornece uma série de atributos que definem o vocabulário de um estilo. Dentre as restrições definidas em Xtyle, estão as regras de conexão entre componentes como *push* e *pull*, representadas nas Figuras 6a e 6b, respectivamente.

Figura 6. Representação de Conexões push (a) e conexões Pull (b)

5. A FERRAMENTA GUIDRAX

Para facilitar o desenvimento de sistemas distribuídos baseados em arquiteturas com infraestrutura de middleware, como CORBA e RMI, foi criado o toolkit DraX. E para ajudar na construção e facilitar a visualização da estrutura proposta em DraX foi criada a ferramenta GUIDraX.

Para melhor apresentação de GUIDraX, dividiu-se a ferramenta em: estrutura interna, ambiente e construção e visualização dos componentes.

5.1 Estrutura Interna
Optou-se por utilizar a linguagem Java para o desenvolvimento da ferramenta GUIDraX, por, dentre seus vários atributos, ser uma linguagem independente de plataforma e grande aceitação na comunidade de desenvolvedores, além de prover abstração por Orientação a Objetos. Logo, neste tópico, tenta-se ilustrar ao leitor a hierarquia de classes que serviram como base para a implementação da ferramenta.

O projeto inicial de GUIDraX prevê a distribuição das classes que implementam suas funcionalidades em 3 pacotes, são eles:

br.cefetce.nash.guidrax
br.cefetce.nash.guidrax.visual
br.cefetce.nash.guidrax.drax

Pacote Default (br.cefetce.nash.guidrax)
Neste pacote, foram colocadas as classes referentes ao controle da ferramenta, ou seja, as classes responsáveis pelo fluxo de dados e estado da entrada de dados do usuário. As classes deste pacote se comunicam com as classes dos pacotes br.cefetce.nash.guidrax.drax (definidas em DraX) e br.cefetce.nash.guidrax.visual. Elas descrevem arquitetura, estilo e componente tanto logicamente quanto graficamente, além gerar e ler arquivos ArchML e Xtyle. Por exemplo: Arquitetura, Estilo, Componente, Interface, Porta e Conexao.

Pacote de Visualização (br.cefetce.nash.guidrax.visual)
Este pacote contém as classes usadas para desenhar o layout da ferramenta. Ele está dividido conceitualmente em:

Classes da Área de Trabalho (AT), por exemplo, ATComponentes, ATArquiteturas, ATEstilos;
Classes da Barra de Ferramentas (BF), por exemplo, BFControle, BFArquiteturas, BFComponentes;
Classes do Painel de Propriedades (PP), por exemplo, PPArquiteturas, PPComponetes, PPEstilos;
Classes do Painel de Controle (PC), por exemplo, PCProjetos.

Pacote de serviços do framework DraX (br.cefetce.nash.guidrax.drax)
Onde estão as classes responsáveis por geração de código CORBA e RMI a partir das especificações de arquitetura, estilo e componentes devidamente validadas.

5.2 Ambiente
De uma maneira mais abrangente, podemos dividir a ferramenta nas seguintes seções (Figura 7):

- Área de Trabalho: Espaço reservado para definição gráfica de novos componentes, arquiteturas e estilos. Também é possível visualizar os códigos gerados em ArchML e Xtyle a partir da mudança de contexto definida na Barra de Ferramentas.
- Barra de Ferramentas: Neste espaço, encontram-se todos os botões referentes a ações (salvar projeto, copiar, novo componente, nova porta, visualizar códigos e etc).
- Painel de Propriedades: Painel responsável por exibir os detalhes das propriedades dos objetos selecionados na Área de Trabalho.
- Painel de Controle: Painel que contém as informações sobre os projetos.

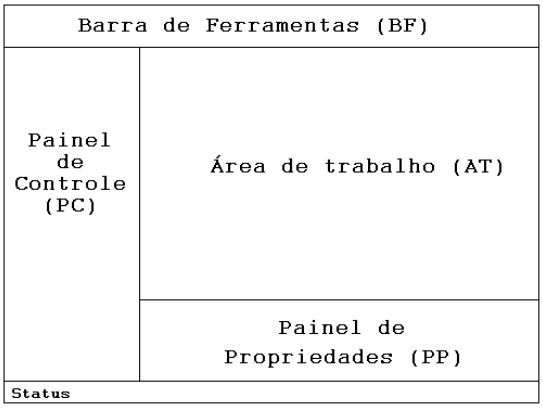


Figura 7. Organização da Ferramenta

5.3 Construção e Visualização dos Componentes na Ferramenta

Em GUIDraX, assim como em DraX, foi estabelecida uma metodologia para o desenvolvimento de aplicações distribuídas, utilizando conceitos de arquitetura de software.

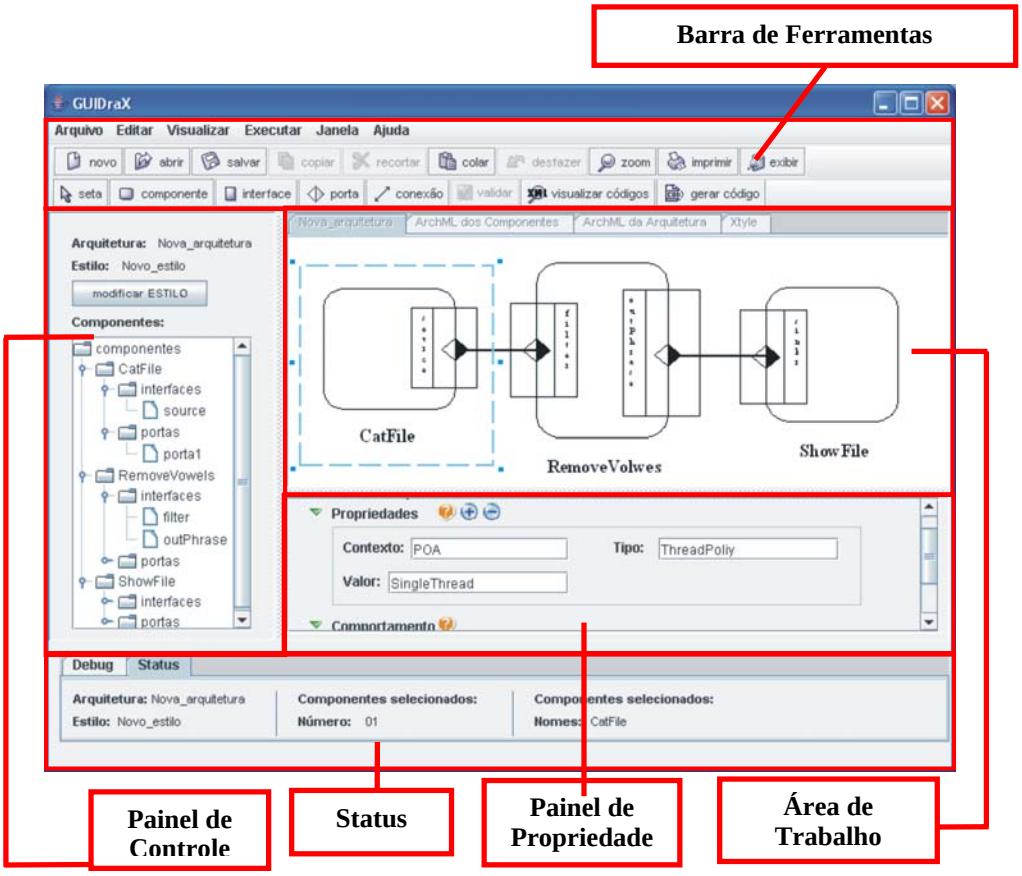


Figura 8. Interface GUIDraX

Declaração da Arquitetura e Especificação do Estilo
Nessa etapa, o desenvolvedor informa os dados básicos necessários para a criação de uma nova arquitetura: nome e estilo associado. Opcionalmente, pode-se criar um novo estilo a ser associado à arquitetura. Também é possível fazer uma descrição informal, simples e funcional da aplicação. Vale ressaltar, que nessa etapa, a definição da arquitetura não é possível de se realizar, uma vez que a mesma depende dos componentes, definidos na próxima etapa.

Especificação dos Componentes
Especificação dos componentes, definindo interfaces e portas, e posterior validação dos mesmos.

Especificação da Arquitetura e Estilo
Utilizando os componentes especificados na etapa anterior, constrói-se a arquitetura como um todo, estabelecendo as conexões entre os componentes. A qualquer momento desta etapa, o estilo associado à arquitetura poderá ser modificado ou trocado por outro estilo.

Geração de Código
Uma vez terminada e validada, uma arquitetura poderá servir como base para a geração de código dos serviços e invocação. Para tal, deve ser escolhido o middleware ao qual se quer gerar o código base para a implementação da lógica da aplicação.

6. ESTUDO DE CASO

Para exemplificar a utilização de GUIDraX, apresentamos nesta seção um a construção de uma aplicação distribuída simples. A aplicação em questão é um sistema de drenagem de uma mina de carvão, cuja especificação está descrita em [9]. Esse sistema foi desenvolvido para controlar o bombeamento, para a superfície, da água acumulada em um poço coletor existente dentro de uma mina de carvão. A Figura 9 mostra a arquitetura do sistema proposto.

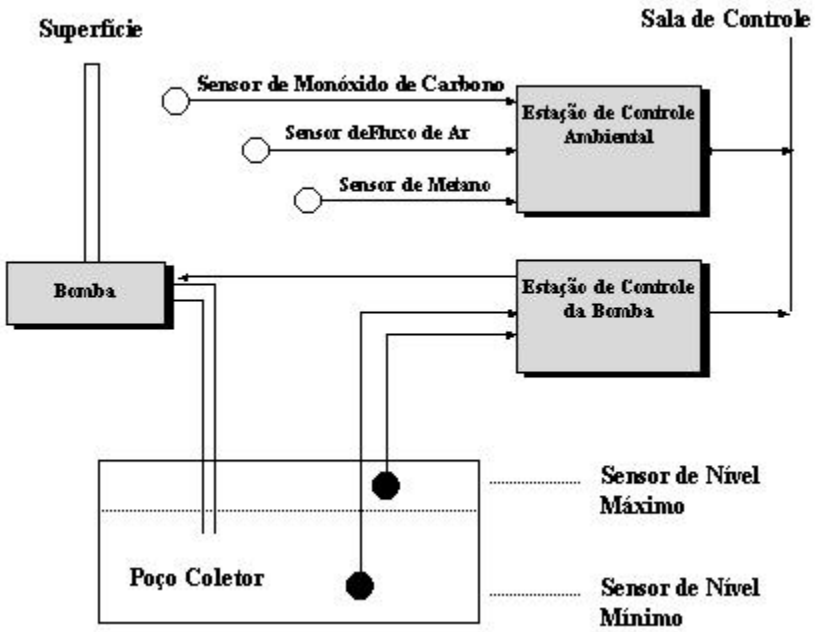


Figura 9. Sistema de Mineração

‘ A bomba é ativada pelo operador na superfície, e trabalha automaticamente, controlada pelo nível de água detectado pelos sensores de nível máximo e de nível mínimo. Se for detectado nível máximo, a bomba é ativada até que o nível mínimo seja atingido. O operador pode desativar a bomba a partir da superfície, e também pode solicitar o estado corrente da bomba.

A bomba fica situada no fundo da mina e, por razões de segurança, não deve ser ligada ou continuar funcionando quando a porcentagem de metano excede um certo limite de segurança. O sistema obtém informações sobre o nível de metano através da estação de monitoramento ambiental, que também monitora o fluxo de ar e o nível de monóxido de carbono dentro da mina. Essas informações são passadas à estação de controle e para a sala de controle na superfície de modo a se evitar situações de risco.

Pela Figura 9, fica fácil perceber que o sistema é decomposto em três módulos principais que representam, a estação de controle da bomba, a estação de monitoramento ambiental e a sala de controle da superfície. Na Figura 10, apresentamos a modelagem do sistema realizada pela ferramenta GUIDraX.

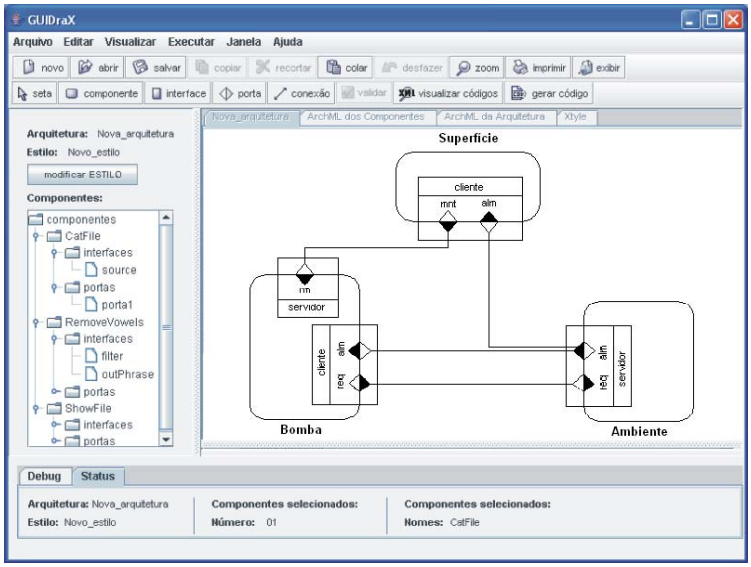


Figura 10. Especificação em GUIDraX

Podemos observar, através das representações visuais da arquitetura (Figura 10), que foi escolhido o estilo Cliente-Servidor para essa aplicação e que as comunicações foram todas definidas síncronas. Dessa forma, a geração de código, cujas propriedades da aplicação foram definidas como sendo baseadas em um middleware CORBA, fornecerá um conjunto de códigos Java que incluirá todas as decisões arquiteturais descritas em GUIDraX. Para exemplificar, a Figura 11 apresenta um trecho de código do componente Bomba.

```
package Mineracao;
public class Bomba extends Mineracao_IDL.BombaPOA{
 static private org.omg.CORBA.ORB orb = null;
 static private org.omg.CosNaming.NamingContext ncRef = null;
 static private org.omg.PortableServer.POA rootpoa = null;
 static private org.omg.CORBA.Object BombaRef = null;
 public static void main(String[] args){
  Bomba BombaObj = new Bomba(args);
   try{
   rootpoa = org.omg.PortableServer.POAHelper.narrow
             (orb.resolve_initial_references ("RootPOA"));
  }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){
         System.exit(1);
      }
...
```

Figura 11. Exemplo de Trecho de Código Gerado.

7. CONCLUSÃO

Apresentamos neste trabalho um ambiente de desenvolvimento visual de aplicações distribuídas. Esse ambiente, denominado GUIDraX, além de facilitar a utilização dos conceitos arquiteturas de software e estilos arquiteturais, propicia a construção de aplicações distribuídas complexas sem a necessidade do domínio de especificação como CORBA e RMI. Para isso, GUIDraX oferece ao usuário uma gramática de elementos gráficos para a utilização das estruturas do ambiente, permitindo a fácil manipulação de conceitos como portas, componentes e sistemas.

Para dar suporte à validação de estruturas de aplicações complexas, estamos desenvolvendo, atualmente, um sistema de validação sintática de aplicações. Assim, um conjunto de validações estruturais poderá ser realizado nas aplicações antes da geração de seus códigos.

Dessa forma, acreditamos que com a utilização de GUIDraX os desenvolvedores terão uma ferramenta completa que permitirá a construção e validação de software distribuído de forma tão amigável quanto as atuais aplicações de desktop.

Referências

[1] Allen, R. and Garlan, D. A Formal Basis for Architectural Connections. *IEEE Transactions on Software Engineering*, 1997.

[2] Bass, L. and Clements, P. and Kazman, R. *Software Architecture in Practice*. Addison-Wesley, 1998.

[3] Dashofy, Eric M. and van der Hoek, A. and Taylor, Richard N. An Infrastructure for the Rapid Development of XML-based Architecture Description Languages. In *Proceedings of the 24th International Conference on Software Engineering (ICSE2002)*, 2002.

[4] Di Nitto, Elisabetta and Rosenblum, David S. Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures. In *International Conference on Software Engineering*, pages 13–22, 1999.

[5] Emmrerich, W. Software Engineering and Middleware: A Roadmap. *The Future of Software Engineering. ACM Press*, 2002.

[6] Garlan, D. Style-Based Refinement for Software Architecture. In Joint Proceedings of the Second International Software Architecture Workshop (ISAW2) and *the International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*. ACM Press, 1996.

[7] Garlan, D. and Kompanek. A. and Melton, R. and Monroe, R. Architectural Style: An Object-Oriented Approach. Technical report, Carnegie Mellon University, 1996.]

[8] Garlan, D and Allen, A. and Ockerbloom, J. Exploiting Style in Architectural Design Environments. *Proceedings of SIGSOFT '94 Symposium on the Foundations of Software Engineerng*, December 1994.

[9] Kramer, J., Magee, J., Sloman, M., Lister,A.: "CONIC: AN INTEGRATED APPROACH TO DISTRIBUTED COMPUTER COLTROL SYSTEMS", *IEEE Proceedings* Part E, Vol. 130, no.1, Janeiro 1983.

[10] Object Management Group. The Commom Object Request Broker: Architecture and Specification Revision 2.4. In *OMG Technical Document formal/00-11-07*, 2000.

[11] Java Soft. Remote Method Invocation (RMI). Disponível em: <http://java.sun.com/products/jdk/rmi/>. Acesso em: 04/03/2006.

[12] Magee, J. and Dulay, N. and Eisenbach, S. and Kramer, J. Specifying Distributed Software Architectures. In *Proceedings of the 5th European Software Engineering Conference*, 1995.

[13] Medvidovic, N. and Taylor, R. N. Architecture Description Languages. In Software Engeneering ESEC/FSE'97, 1997.

[14] Shaw, M. and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.

[15] Souza, Cidcley T. de and Cunha, Paulo. R. F. Descrição de Arquiteturas e Estilos em DraX. *Relatório Técnico. Universidade Federal de Pernambuco*, 2003.