Uma Arquitetura Reflexiva Baseada na Web para Ambiente de Suporte a Processo[1]

Marcelo Hideki Yamaguti, M.Sc.[*,**]
Roberto Tom Price, D.Phil.[*]

[*] Universidade Federal do Rio Grande do Sul - Instituto de Informática
Porto Alegre - RS - E-mail: {yama,tomprice}@inf.ufrgs.br

[**] Pontifícia Universidade Católica do Rio Grande do Sul - Faculdade de Informática
Porto Alegre - RS - E-mail: yama@inf.pucrs.br

Resumo

Este trabalho apresenta uma arquitetura reflexiva baseada na Web, nomeada WRAPPER (Web-based Reflective Architecture for Process suPport EnviRonment), cujo objetivo principal é prover uma infra-estrutura para um ambiente de suporte a processo de software. A motivação principal para esta arquitetura vem da necessidade de se obter maior flexibilidade na gerência de processo de software. Esta flexibilidade é obtida através do uso de mecanismos de reflexão computacional. A arquitetura usa a World Wide Web como sua plataforma e provê mecanismos para definir e controlar objetos (e meta-objetos) distribuídos sobre um middleware CORBA. Como resultado, a arquitetura provê: flexibilidade na gerência de processo, permitindo o controle e adaptação do mesmo; distribuição na Web, permitindo a distribuição de tarefas do processo e o uso de ferramentas do ambiente em locais remotos; e heterogeneidade para agregar, ao ambiente, ferramentas de plataformas e fornecedores diversos. Neste contexto, este trabalho apresenta a estrutura da arquitetura reflexiva, os mecanismos usados para a modelagem e execução de processo no ambiente, bem como o protótipo implementado do ambiente de suporte a processo de software. ***Palavras-chaves****: ambiente de desenvolvimento de software, ambiente de suporte a processo, reflexão computacional, arquitetura reflexiva, World Wide Web.*

Abstract

This paper presents a web-based reflective architecture, named WRAPPER (Web-based Reflective Architecture for Process suPport EnviRonment), whose main goal is to provide an infrastructure for a software process support environment. The main motivation for this architecture comes from the need of obtaining more flexibility in the software process management. This flexibility is obtained through the use of computational reflection mechanisms. The architecture uses the World Wide Web as its platform and it provides mechanisms to define and control distributed objects (and meta-objects) on a CORBA middleware. As result, the architecture provides: flexibility in process management, allowing the control and the adaptation of the process; distribution in the Web, allowing the distribution of process tasks and the use of the environment tools in remote locations; and heterogeneity to aggregate, to the environment, tools from different platforms and suppliers. In this context, this work presents the structure of the reflective architecture, the applied mechanisms for modeling and executing the process in the environment, as well as, the implemented prototype of the software process support environment. ***Keywords****: software development environment, process support environment, computational reflection, reflective architecture, World Wide Web.*

1. Introdução

Um ADS (Ambiente de Desenvolvimento de *Software*) busca integrar diferentes ferramentas CASE (*Computer Aided Software Engineering*) para prover um suporte computacional adequado ao desenvolvimento de *software*. Para obter uma integração conveniente alguns aspectos devem ser observados. Um destes aspectos é a integração de processo de *software*. Esta integração é obtida quando as ferramentas do ADS suportam um

[1] Parcialmente financiado pelo CNPq.

processo comum de desenvolvimento de *software*, neste caso o ADS pode ser chamado de PSEE (*Process-centered Software Engineering Environment*).

A construção de PSEEs levantam algumas questões e problemas. Em especial, para suportar um processo de *software*, o PSEE deve definir formas para especificar, executar e controlar este processo. Uma possível alternativa seria definir a arquitetura do PSEE baseada em objetos reflexivos. Estes objetos permitiriam obter maior flexibilidade no controle do processo.

Em uma arquitetura reflexiva, baseada em objetos, a estrutura é dividida basicamente em dois níveis: o nível base que é constituído pelos objetos que implementam os aspectos funcionais do *software*, e o meta-nível que possui meta-objetos que implementam aspectos não-funcionais e que podem inspecionar e mesmo alterar os objetos do nível base.

Em um PSEE baseado nesta arquitetura, o processo de *software* é construído como um conjunto de classes e objetos que representam os elementos do processo. Além disso, as ferramentas integradas ao ambiente também são tratadas como objetos. Todos estes objetos constituem o nível base da arquitetura. Através dos meta-objetos, no meta-nível, é possível controlar, estender e alterar o processo de *software* e mesmo o próprio ambiente. Esta é a abordagem da arquitetura proposta neste trabalho, chamada WRAPPER (*Web-based Reflective Architecture for Process suPport EnviRonment*) - Arquitetura Reflexiva baseada na Web para Ambiente de Suporte a Processo.

O protótipo desta arquitetura está implementado com objetos (e meta-objetos) distribuídos em um *middleware* CORBA, usando a *World Wide Web* como plataforma. Como resultado, a arquitetura provê: flexibilidade para o controle e adaptação do processo e do ambiente, através do uso de objetos reflexivos; distribuição do ambiente permitindo a distribuição de atividades do processo e utilização de ferramentas do ambiente em locais remotos, através do uso de objetos distribuídos e tecnologias Web; e heterogeneidade para agregar ao ambiente ferramentas de diversas plataformas e fornecedores, através do uso CORBA e de tecnologias Web.

Este trabalho está estruturado da seguinte forma: na seção 2, discute-se os aspectos básicos de PSEEs. Na seção 3 discute-se o uso de objetos para se definir um processo; na seção 4 apresenta-se os conceitos de objetos reflexivos. Na seção 5, a arquitetura reflexiva proposta para suporte a processo é apresentada. Na seção 6, é apresentado o protótipo implementado. As conclusões são apresentadas na seção 7.

2. Ambiente de Desenvolvimento de *Software* Centrado em Processo

Em Engenharia de *Software*, existem diversas maneiras de se obter maior produtividade na construção de sistemas com melhor qualidade. A automação do desenvolvimento de *software* pelo uso de ferramentas CASE é um destes meios.

Uma ferramenta CASE isolada pode prover suporte para uma ou mais atividades envolvidas em um desenvolvimento de *software*, mas o real poder da tecnologia CASE surge quando diversas ferramentas CASE trabalham de forma integrada. Um ADS [24] é um *framework* que integra diferentes ferramentas CASE, com o objetivo de prover suporte para um processo completo de desenvolvimento de *software* (ou de grande parte deste).

Desta maneira, diferentes ferramentas podem ser usadas em diferentes atividades, provavelmente por diferentes pessoas, para o desenvolvimento de *software* de uma maneira integrada. Para obter esta integração, alguns aspectos devem ser considerados. Um destes aspectos é a integração de processo. Este tipo de integração supõe que as ferramentas no ambiente suportam um processo de *software* comum [25].

O interesse no processo de *software* vem da percepção que para obter um *software* (o produto final com todos os artefatos relacionados, tais como especificações, códigos,

documentação, resultados de testes, etc.) de boa qualidade é necessário executar com alta qualidade cada uma das atividades envolvidas na produção do *software* [10][12].

Um PSEE é um ADS que provê suporte computacional para permitir a especificação, execução e controle de um processo de *software*.

A construção de um PSEE, como qualquer ADS, envolve alguns problemas, tais como:

- como agregar novas ferramentas CASE ao ambiente?
- como gerenciar a distribuição de ferramentas?

Além disso, o suporte a um processo traz outras questões:

- como especificar o modelo do processo?
- como adaptar o processo?
- como suportar diferentes processos?
- como definir os artefatos, ferramentas e agentes específicos envolvidos na execução do processo?
- como controlar a execução do processo?
- como obter métricas e estatísticas sobre esta execução?

Diferentes ambientes [2][10][11][19] propõem soluções diversas para estas questões. A solução proposta pela arquitetura WRAPPER é baseada no uso de reflexão computacional através de objetos distribuídos e reflexivos.

3. Modelando processo com objetos

Este trabalho supõe que um processo de *software* pode ser definido como um conjunto de classes e objetos computacionais. O uso de orientação a objetos em processo de *software* influenciou diversos ambientes e trabalhos [8][10]. Usando uma visão computacional orientada a objetos, os conceitos básicos de processo de *software* são mapeados da seguinte maneira:

- Linguagem de modelagem de processo: é definida por uma linguagem de modelagem de objetos. Este trabalho utiliza a UML (*Unified Modeling Language*) [3] para este propósito. UML é usada pois está se tornando um padrão *de facto* para a modelagem de objetos e porque provê mecanismos para estendê-la. Um diagrama de classes é utilizado para modelar as meta-classes envolvidas em processos genéricos de *software*. A figura 1 apresenta um meta-modelo que representa os elementos básicos envolvidos em processos genéricos (meta-processos) de *software* [8][10].

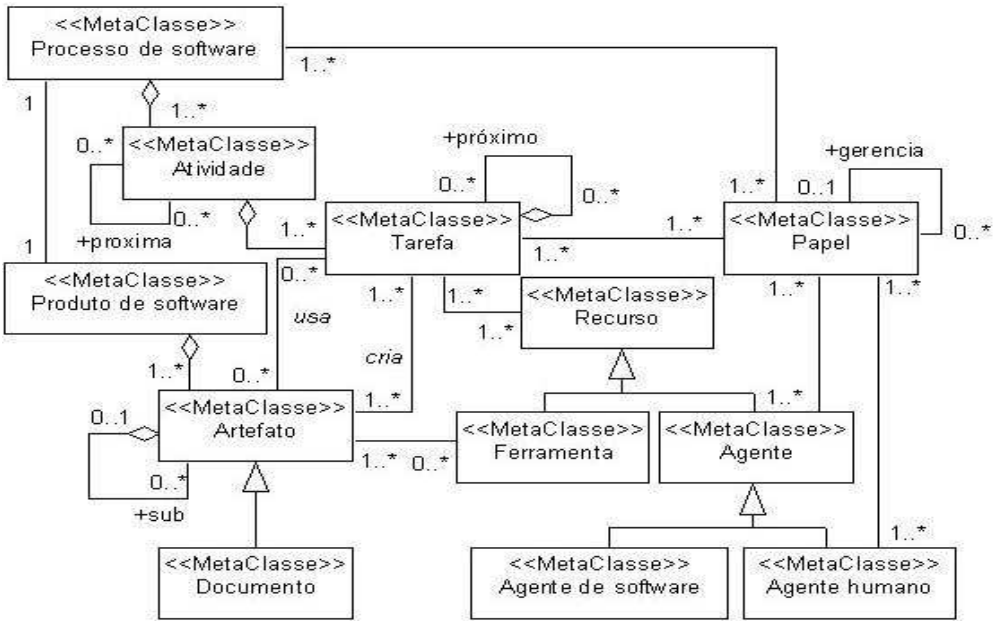

Figura 1: meta-modelo de meta-processos.

- Meta-processo: é definido por um modelo que representa as classes instanciadas para um processo de *software* particular. Este modelo é provido por um diagrama de classes UML que mostra as classes básicas deste processo. A figura 2 mostra um diagrama (parcial) que define as classes envolvidas no Processo Unificado (adaptado de [13]) - atividade de análise.

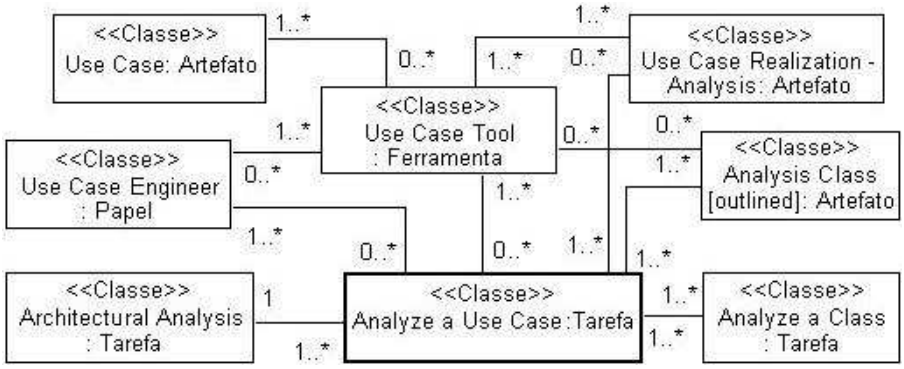

Figura 2: um modelo de meta-processo - parcial.

- Projeto: é definido por um conjunto de objetos instanciados a partir do modelo de um meta-processo. O projeto é representado por um diagrama de objetos UML. Cada objeto representa um elemento (por exemplo, uma tarefa, uma ferramenta, um agente) alocado e envolvido em um projeto de *software*. Adicionalmente, o modelo de *workflow* do projeto também deve ser gerado. Este modelo é representado por um diagrama de atividades UML que conecta as tarefas do projeto, bem como, os demais objetos envolvidos no mesmo.

- Execução do processo: a execução do processo corresponde a execução do *workflow* correspondente. Neste trabalho é pressuposto que a execução do projeto é controlada por um sistema de gerência de *workflow* - WfMS (*Workflow Management System*) [1][28] - que ativa os objetos do projeto de acordo com o modelo de *workflow* definido.

4. Objetos reflexivos

Um sistema é dito ser reflexivo se é capaz de processar seu próprio processamento [17]. Especificamente, um sistema é reflexivo se pode obter dados sobre sua própria computação e se pode atuar e mudar o seu próprio comportamento durante sua execução.

A reflexão computacional está presente em muitos paradigmas de linguagens de programação. Entretanto, este conceito é melhor explorado no paradigma de orientação a objetos. Basicamente um sistema reflexivo orientado a objetos possui objetos organizados em dois níveis. No nível base ficam os objetos que implementam a execução normal (requisitos funcionais) do sistema. No meta-nível ficam os objetos (meta-objetos) que podem implementar requisitos não-funcionais (tais como segurança ou tolerância a falhas) do sistema. Um MOP (*Meta-Object Protocol*) define como e quando o meta-nível é associado ao nível base. Um MOP pode definir que um meta-objeto fique associado a somente um objeto, a um grupo de objetos, a um único método, a uma classe ou quaisquer outras possíveis combinações das características do nível base.

Denomina-se reflexão estrutural quando o meta-nível consegue apenas manipular a estrutura estática das classes de objetos do nível base; ao passo que na reflexão comportamental o meta-nível consegue manipular o comportamento dos objetos do nível base. Na reflexão comportamental, um meta-objeto é ativado quando o elemento de nível base associado recebe uma mensagem. O meta-objeto pode, então, inspecionar o elemento, executar alguma computação adicional, ou mesmo ativar outro elemento antes de passar (ou não) a mensagem adiante.

O uso de reflexão computacional no desenvolvimento de *software* traz um grande benefício [5][16]: a adaptabilidade. Através do meta-nível é possível monitorar, estender ou mesmo alterar o funcionamento do nível base, sem que este tenha que ser re-implementado.

Algumas linguagens orientadas a objetos foram criadas ou adaptadas para suportar reflexão [5]. Smalltalk, CLOS, e Open C++ são exemplos de tais linguagens. Além de linguagens, arquiteturas baseadas em reflexão computacional também foram desenvolvidas. Alguns exemplos de arquiteturas reflexivas são: Guaraná [21] (baseada em Java), Luthier [6] (baseada em Smalltalk) e MOTF [15] (baseada em Open C++). A arquitetura WRAPPER é uma arquitetura reflexiva baseada no *middleware* CORBA.

4.1. Reflexão em CORBA

CORBA (*Common Object Request Broker Architecture*) [22] é um *middleware* que permite realizar a comunicação e a computação de objetos distribuídos e heterogêneos (implantados em diferentes linguagens de programação e plataformas computacionais).

Um objeto em CORBA torna-se disponível para outros quando é registrado em um ORB (*Object Request Broker*). O registro do objeto é realizado pelo processamento da sua especificação IDL (*Interface Definition Language*). Uma especificação IDL declara a interface do objeto e pode ser armazenada em um IR (*Interface Repository*) permitindo que outros objetos possam dinamicamente acessar a interface de objetos registrados no ORB. Através do protocolo IIOP (*Internet InterOrb Protocol*), diferentes ORBs podem se comunicar através da Web.

Em termos de reflexão computacional, CORBA provê mecanismos que podem ser explorados para a reflexão estrutural e comportamental. Para a reflexão estrutural, CORBA provê facilidades de manipulação das definições IDL registradas no IR. O IR é parte da arquitetura CORBA e atua como um banco de dados *on-line* que provê acesso as definições de interfaces de objetos especificadas em IDL. Usando o IR é possível dinamicamente manipular a estrutura estática de objetos registrados no ORB (reflexão estrutural).

Em termos de reflexão comportamental, pode-se explorar uma característica especial de CORBA: *interceptors*. Eles são parte da especificação CORBA [23] e constituem-se em uma facilidade prevista para adicionar funcionalidades ao ORB. *Interceptors* são inseridos entre os objetos e o ORB, e são automaticamente ativados quando uma mensagem é enviada ou recebida através do mesmo. Ao ser ativado, um *interceptor* pode analisar a mensagem e/ou o objeto correspondente, executar computação extra (inclusive alterando o comportamento original previsto) e então passar a mensagem adiante (estas características podem ser usadas para a reflexão comportamental).

5. A arquitetura WRAPPER

Baseado nos conceitos apresentados, nesta seção é descrita a arquitetura reflexiva proposta neste trabalho: a arquitetura WRAPPER. Esta arquitetura é organizada em dois níveis básicos: o nível base e o meta nível. Acima destes dois níveis é construído o nível de aplicação que corresponde às ferramentas do ambiente de suporte a processo.

O nível base é representado pelos objetos da aplicação. Neste trabalho, os objetos representam os elementos básicos envolvidos no ambiente de suporte, bem como os elementos envolvidos no processo de *software*. O meta-nível é responsável por prover mecanismos para gerenciar os meta-objetos que monitoram os objetos do nível base.

As subseções seguintes descrevem os níveis da arquitetura em maior detalhe.

5.1. O nível base

O nível base (figura 3) é baseado em um ORB CORBA. O ORB provê facilidades básicas, tais como registro de objetos, processamento de IDL, operações de manipulação do próprio ORB e do IR correspondente. Para intermediar as operações de registro e consulta aos objetos no ORB, a arquitetura WRAPPER possui um gerente de nível base.

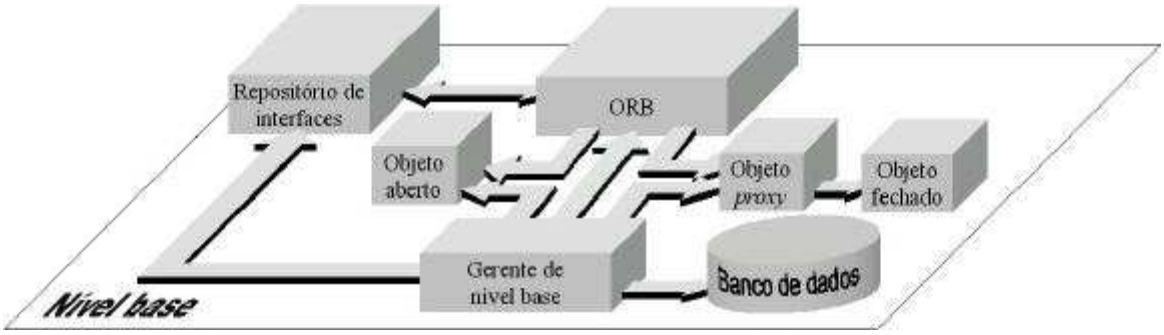

Figura 3: estrutura do nível base.

No registro de um objeto, este pode ser de um dos seguintes tipos:

- Objeto "aberto": é um componente criado para o ambiente ou que provê acesso a sua estrutura interna (por exemplo, uma ferramenta CASE de código aberto), possibilitando a obtenção de informações mais detalhadas sobre o objeto, inclusive a ativação de suas operações.
- Objeto "fechado": representa um componente que não provê acesso a sua estrutura interna (por exemplo, ferramentas CASE proprietárias ou *software* legado). Portanto, as informações sobre o objeto ficam restritas à sua ativação.

Um objeto "aberto" é registrado pelo gerente através da definição e processamento de sua especificação IDL que é armazenada no IR. Para o registro de um objeto "fechado" o gerente cria e registra um objeto-*proxy* correspondente, que fica então responsável pelo tratamento de chamadas ao objeto "fechado". O objeto-*proxy* é tratado como um objeto CORBA comum, porém suas informações e comportamento se limitam a ativação do objeto "fechado". Um banco de dados é utilizado pelo gerente para armazenar os objetos (abertos e *proxy*) persistentes neste nível.

Além do controle de registros, o gerente de nível base é responsável por prover facilidades aos níveis superiores para manipular os objetos registrados no ORB. O gerente de nível base também é tratado como um objeto CORBA e possui sua interface especificada em IDL e disponibilizada no IR. Desta forma, qualquer objeto pode acessar este gerente e consultar os objetos registrados no ORB através das suas operações disponibilizadas.

O nível base é executado em um ORB-servidor e nos ORBs-clientes. O ORB-servidor é o que disponibiliza um IR e um banco de dados. Neste ORB também ficam localizados os objetos envolvidos no *workflow* de processo, bem como os objetos utilizados pelos ORBs clientes (como ferramentas CASE abertas).

Um ORB-cliente pode ser de dois tipos: um ORB completamente instalado no cliente, ou um ORB em execução dentro de um navegador Web. Um ORB-cliente completamente instalado possui funcionalidade completa (inclusive seu próprio IR e banco de dados), exceto por não possuir os objetos de *workflow* de processo (que ficam centralizados no ORB servidor). Um ORB-cliente dentro de um navegador Web tem suas funcionalidades limitadas pelo ambiente em que estiver executando. Caso o ambiente permita, objetos-*proxy* podem ser criados e mesmo gravados em arquivos locais, porém este ORB-cliente não provê facilidades como um IR próprio, nem banco de dados para objetos persistentes.

5.2. O meta-nível

O meta-nível desta arquitetura controla os meta-objetos conectados a objetos do nível base (figura 4).

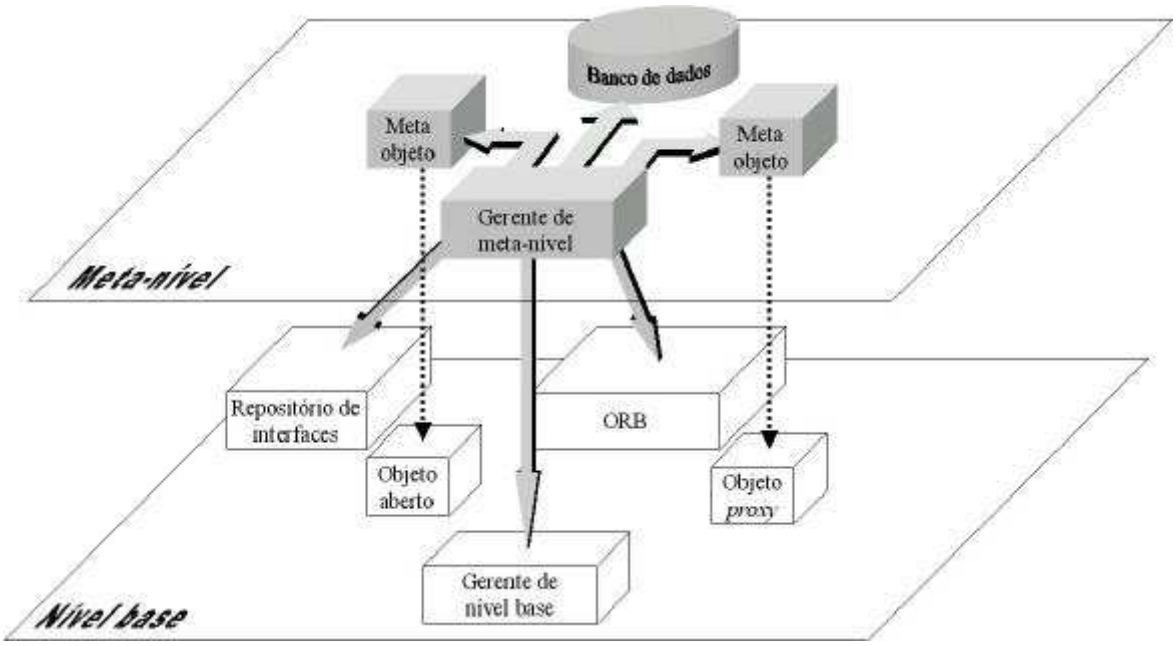

Figura 4: estrutura do meta-nível.

A criação e registro de meta-objetos no ORB são intermediados por um gerente de meta-nível. O comportamento de um meta-objeto, depende de seu objetivo. Um meta-objeto pode, por exemplo, coletar dados estatísticos (consultando informações do objeto associado) ou adicionar funcionalidades ao ambiente (alterando a execução do objeto associado, pela execução de computações adicionais ou pela chamada de outros objetos).

Na criação de um meta-objeto, o gerente de meta-nível acessa o gerente de nível base para consultar os objetos do nível base registrados no ORB. Dependendo do objetivo do meta-objeto este pode ser associado de diversas maneiras ao nível base, o MOP (*Meta-Object Protocol*) escolhido define esta associação.

O meta-objeto também é tratado como um objeto CORBA e também é registrado no ORB. Desta forma, é possível criar um meta-objeto sobre um meta-objeto já existente, criando níveis de reflexão mais altos (conceito de "torre de reflexão" [16]).

Dados coletados por meta-objetos podem ser armazenados em um banco de dados em separado. O gerente de meta-nível também é responsável por prover uma interface de manipulação deste banco de dados, permitindo que as informações armazenadas possam ser usadas por gerentes humanos ou por outros objetos do ambiente.

Na arquitetura WRAPPER um meta-objeto é implementado por um *interceptor* CORBA. O gerente de meta-nível é o responsável por controlar e registrar os *interceptors* no ORB. Cada interceptor fica responsável por interceptar e tratar as mensagens do elemento de nível base associado.

5.3. O nível de aplicação

Sobre os dois níveis apresentados anteriormente, a arquitetura define o nível de aplicação que provê a interface de uso do ambiente para o suporte a processo de *software*. Este nível (figura 5) é implementado por aplicações, no servidor e nos clientes, que interagem com o ORB e com os gerentes dos níveis inferiores.

Algumas aplicações simplesmente provêem interface gráfica para a manipulação dos gerentes do meta-nível e do nível base. Ferramentas CASE disponíveis pelo ambiente também

fazem parte deste nível. Estas ferramentas provêem diversas funcionalidades como a agregação de outras ferramentas ao ambiente, acesso ao banco de dados do meta-nível, controle de *log* no ambiente, etc.

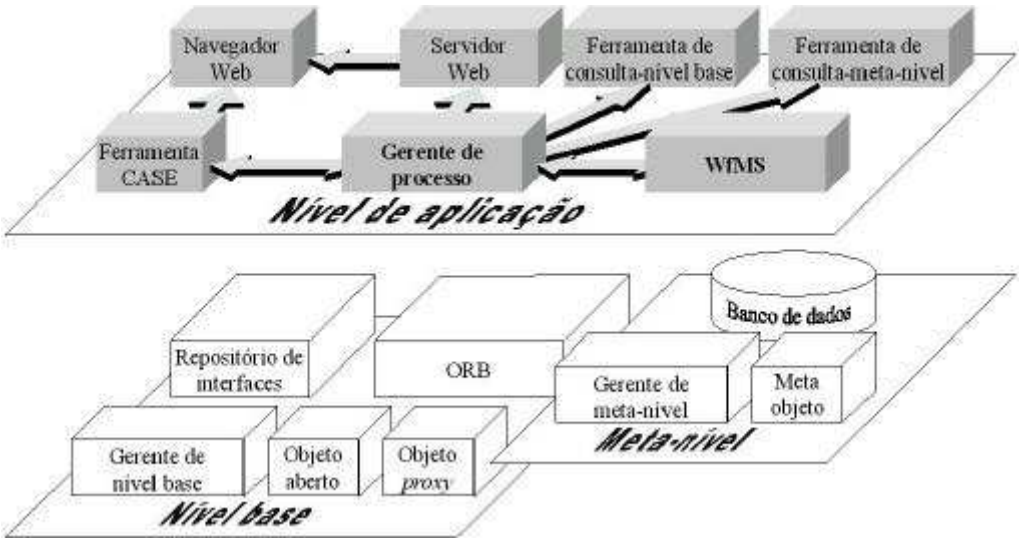

Figura 5: nível de aplicação.

Para permitir o acesso remoto do ambiente é utilizado (no servidor) um *Web Server* que disponibiliza páginas HTML que são acessadas (nos clientes) através de navegadores Web.

Os componentes principais neste nível são o gerente de processo e o sistema de gerência de *workflow* (WfMS). O gerente de processo é uma aplicação que permite executar as operações básicas de gerenciamento do processo, tais como: modelagem, execução e controle do processo de *software*. O WfMS é utilizado para controlar a execução de um projeto de *software* instanciado de um determinado processo de *software*.

Ferramentas CASE diversas são ativadas pelo gerente de processo para as atividades básicas de gerenciamento do processo:

- modelagem do meta-modelo de meta-processos: permite a manipulação do meta-modelo básico de processos genéricos.
- modelagem de um meta-modelo de um processo: permite a definição do meta-modelo de um processo de *software* específico, que é instanciado a partir do meta-modelo de meta-processos.
- modelagem de um projeto: permite a definição de um projeto de *software* instanciado a partir de um meta-modelo de um processo específico. A modelagem e a execução de um projeto de software é apresentada abaixo.

Para a modelagem de um projeto de *software*, inicialmente é escolhido um processo de *software* existente (a partir dos meta-modelos de processos existentes). A partir do meta-modelo (classes) do processo desejado são instanciados os objetos correspondentes.

A instanciação destes objetos implica em relacionar objetos existentes do ambiente (ferramentas CASE, agentes, etc.) ao modelo. Uma parte importante desta instanciação é a definição das diversas tarefas envolvidas no projeto de *software*.

A modelagem do projeto inclui também a modelagem do *workflow* do mesmo. Este modelo conecta e ordena as diversas tarefas definidas. Além das tarefas, o modelo de *workflow* também relaciona os demais objetos instanciados. Por exemplo: para uma determinada tarefa, pode-se relacionar um artefato de entrada, um artefato de saída, uma ferramenta a ser ativada pela tarefa e um agente responsável pela execução da tarefa.

Após a geração do modelo do projeto, este pode ser executado. A execução de um projeto corresponde a execução de seu modelo de *workflow*. O WfMS é o responsável pelo

controle de ativação dos objetos definidos neste modelo. À medida que os objetos do nível base (contidos no modelo de *workflow*) são ativados e estes trocam mensagens entre si.

Para permitir que as tarefas do projeto sejam distribuídas em locais remotos, usando a Web como plataforma, os objetos correspondentes às tarefas apresentam e direcionam a sua execução através de páginas hipertexto. Páginas HTML são então disponibilizadas no *Web Server*.

Uma pessoa (no papel de gerente de processo) pode desejar extrair informações sobre a execução do projeto. Uma maneira de se obter isto seria prever, já na modelagem do meta-modelo de processo, algumas tarefas de gerência que extrairiam as informações desejadas. Na instanciação de um projeto, objetos correspondentes seriam criados e inseridos no seu *workflow*.

Outra maneira, pela abordagem proposta neste trabalho, seria o uso de meta-objetos que monitorariam os objetos envolvidos no modelo de *workflow*. Meta-objetos poderiam ser instanciados e conectados a objetos de nível base, durante a modelagem do projeto. Entretanto, como os meta-objetos são definidos independentemente do meta-modelo de processo, estes podem ser criados e conectados aos objetos de *workflow* a qualquer momento, inclusive quando o *workflow* já estiver em execução, permitindo maior flexibilidade na gerência de processo.

Uma vez instanciado e conectado ao nível base, um meta-objeto será ativado sempre que houver trocas de mensagens com o objeto correspondente. Assim, quando o WfMS tentar ativar uma tarefa (que será representada por um objeto de nível base), isto corresponderá ao envio de uma mensagem ao objeto. O ORB poderá então interceptar a mensagem e ativar quaisquer meta-objetos correspondentes, que poderão executar seu comportamento previsto, como por exemplo, coletar dados estatísticos da execução.

Adicionalmente, o uso de meta-objetos permite que um projeto já instanciado e em execução possa ser alterado. Meta-objetos podem interceptar as mensagens dos seus objetos e mudar a execução prevista do *workflow*. Por exemplo: um meta-objeto pode interceptar a chamada de uma ferramenta e desviar a chamada para uma ferramenta diferente, ou pode interceptar a ativação de uma tarefa e chamar uma nova tarefa definida posteriormente. Desta forma, o uso de meta-objetos permitiria a alteração do processo de *software* e mesmo do próprio ambiente de suporte de processo.

6. O protótipo WRAPPER

Este trabalho está sendo implementado em duas frentes: a estrutura básica da arquitetura (nível base e meta-nível) e o ambiente de suporte a processo (nível de aplicação). A estrutura atual do protótipo é apresentada na figura 6.

A implementação da estrutura básica da arquitetura está baseada em Java sobre o ORB Inprise Visibroker, versão 4.5 [4]. Estão sendo utilizados bancos de dados relacionais acessados por uma ponte JDBC-ODBC [26] para o armazenamento de objetos persistentes. O ambiente de suporte a processo também está sendo implementado em Java. O armazenamento e acesso a páginas HTML no servidor provido pelo *Web Server Internet Information Services* (IIS) [18]. O IIS também é utilizado para a obtenção de referência do ORB e permitir a comunicação entre ORBs por IIOP.

O lado do cliente é executado em navegadores Web, utilizando-se das máquinas virtuais Java (JVM - *Java Virtual Machine*) [26] disponíveis nos mesmos. A interface com o usuário é provida basicamente por *applets* Java embutidos em páginas HTML. Nas subseções seguintes, as características e soluções de implementação da arquitetura são apresentadas.

6.1. Nível base

O nível base da arquitetura está sendo implementado em Java sobre o ORB Inprise Visibroker. Este ORB provê uma API (*Application Programming Interface*) Java para acesso às suas operações básicas, facilidades de processamento de IDL e de acesso do IR.

O gerente de nível base está implementado como uma aplicação Java executando no servidor. Este gerente provê operações básicas de registro de novos objetos, consulta e remoção de objetos do ORB, e armazenamento de objetos persistentes no banco de dados relacional.

Objetos "abertos" são registrados no ORB e sua definição IDL correspondente é armazenada no IR. Para o registro de um objeto "fechado" o gerente instancia um objeto-*proxy* que é registrado no ORB. A definição IDL de objetos-*proxy* já está no IR.

A descrição acima representa a visão do lado do servidor da arquitetura. O lado do cliente está implementado através de *applets* em execução em navegadores Web. O cliente também possui um gerente de nível base, porém com algumas limitações: não há acesso a banco de dados local, nem seu próprio IR. Objetos-*proxy* criados no cliente são armazenados em arquivos locais. Um cliente instalado como aplicação completa ainda está em desenvolvimento.

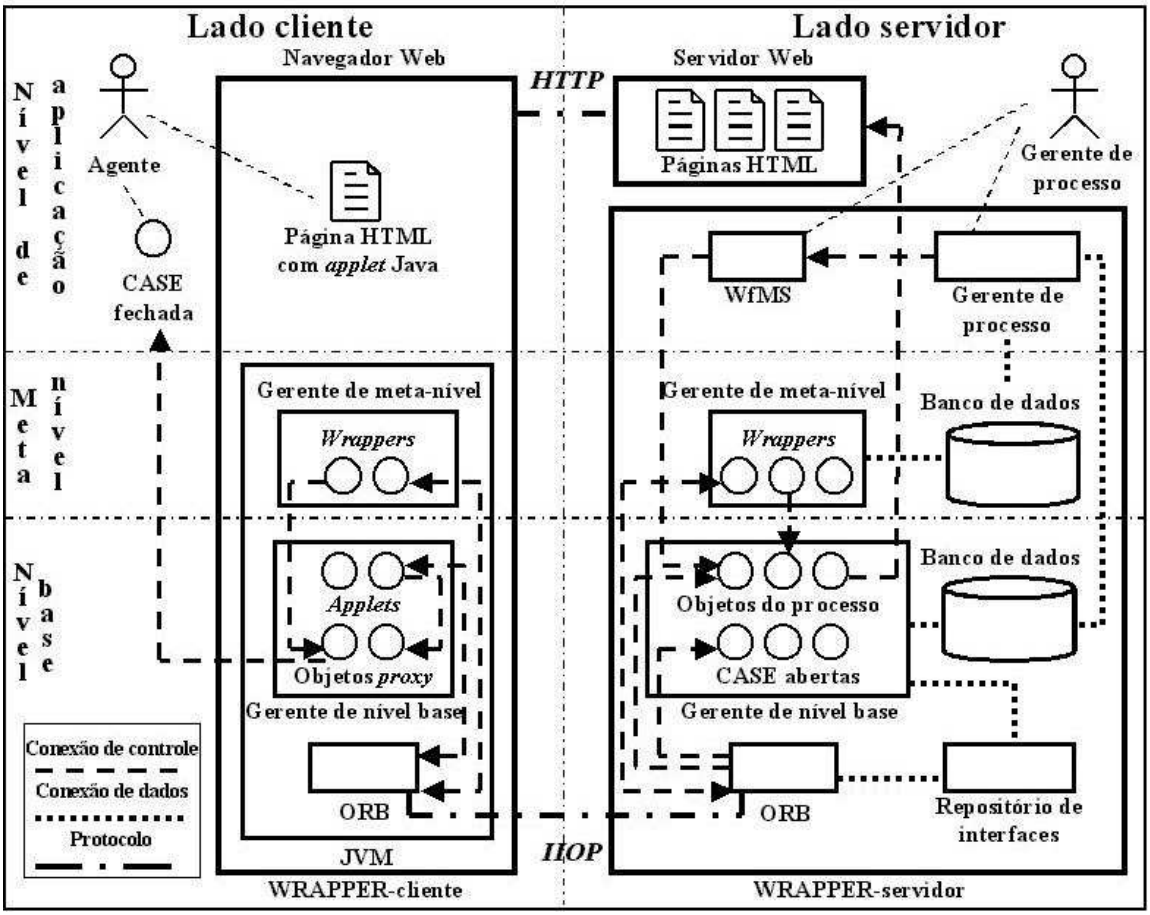

Figura 6: estrutura do protótipo WRAPPER.

6.2. Meta-nível

O meta-nível também utiliza a estrutura do ORB Visibroker. O protótipo implementado utiliza um tipo especial de *interceptor* provido por este ORB, chamado de *wrapper*. Um *wrapper* é um *interceptor* que permite interceptar até mesmo um único método de um objeto.

Wrappers são criados e associados a objetos registrados no ORB. Um *wrapper* é ativado sempre que o objeto ou método associado for chamado. Quando ativado o *wrapper*, atua como um meta-objeto. *Wrappers* podem ser registrados no lado do cliente, ou do servidor, ou em ambos. É possível o encadeamento de diversos *wrappers*, de forma que um *wrapper* possa executar processamento adicional sobre outro *wrapper* (simulando o conceito de "torre de reflexão").

Um *wrapper* no cliente, se necessitar acesso ao banco de dados precisa se comunicar com o gerente de meta-nível do servidor. Se um *wrapper* no servidor necessitar acessar um objeto em um cliente, é utilizado o mecanismo de *callback* [4], que permite que um ORB cliente, atue como um ORB servidor.

O gerente de meta-nível está implementado como duas aplicações Java: o Gerente de *Wrappers* e Serviços *Wrapper*. O Gerente de *Wrappers* é responsável pelo controle de *wrappers* instanciados. A criação de um *wrapper* é intermediada por este gerente que define o MOP de cada *wrapper*.

Os Serviços *Wrapper* provêem serviços básicos que os *wrappers* podem usar, como por exemplo: armazenar informações no banco de dados ou obter acesso e executar um outro objeto do ORB.

6.3. Nível de aplicação

O nível de aplicação está implementado por aplicações Java e *applets* embutidas em páginas HTML no *Web Server*. Os componentes principais: gerente de processo e WfMS estão implementados em Java, permitindo a execução tanto como aplicação ou como *applet*.

O gerente de processo permite a instanciação do diagrama de objetos do modelo de projeto. Existem meta-modelos de processos instanciados predefinidos para permitir a modelagem de projetos. Adicionalmente, o gerente de processo permite ativar ferramentas individuais. O WfMS permite instanciar um diagrama de atividades correspondente ao modelo de *workflow* do projeto e executar este modelo.

Existem diversas ferramentas individuais que provêem interface gráfica a componentes dos níveis inferiores, tais como: visualizador de objetos registrados no ORB, visualizador de interfaces do IR, controlador de *log* de usuário, registrador de ferramentas no ambiente, instanciador de *wrappers*, visualizador de serviços *wrapper* disponíveis, etc.

6.4. Exemplos

Nesta subseção são apresentados alguns cenários de uso do ambiente. Estes foram executados nos navegadores Web Netscape Navigator e Internet Explorer, utilizando-se os sistemas operacionais Linux e Windows 98, tanto em rede local quanto em rede externa ao servidor.

O ambiente no cliente é acessado por um navegador Web. No servidor, o ambiente pode ser acessado, tanto através de aplicações Java, como através de navegador Web. Após o *login*, um usuário recebe uma tela principal mostrando as opções disponíveis para o mesmo. Um gerente de processo pode acessar ferramentas para controlar o *workflow* de um projeto.

Os cenários seguintes exemplificam o uso de meta-objetos (*wrappers*) para extrair informação e para a alteração da execução de um processo.

6.4.1. Cenário 1: coletando dados estatísticos. Neste exemplo um gerente de processo define um meta-objeto sobre um agente, de forma a coletar informações sobre os seus *logins* no ambiente e armazená-los em um arquivo de *log*.

Para realizar isto, quando o projeto está instanciado (diagramas de objetos e atividades realizados) (figura 7), o gerente de processo acessa o WfMS e indica que deseja inserir um

novo *wrapper*. Após escolher o objeto desejado, uma tela de Serviços *Wrapper* é apresentada para escolher a ação desejada para o *wrapper*.

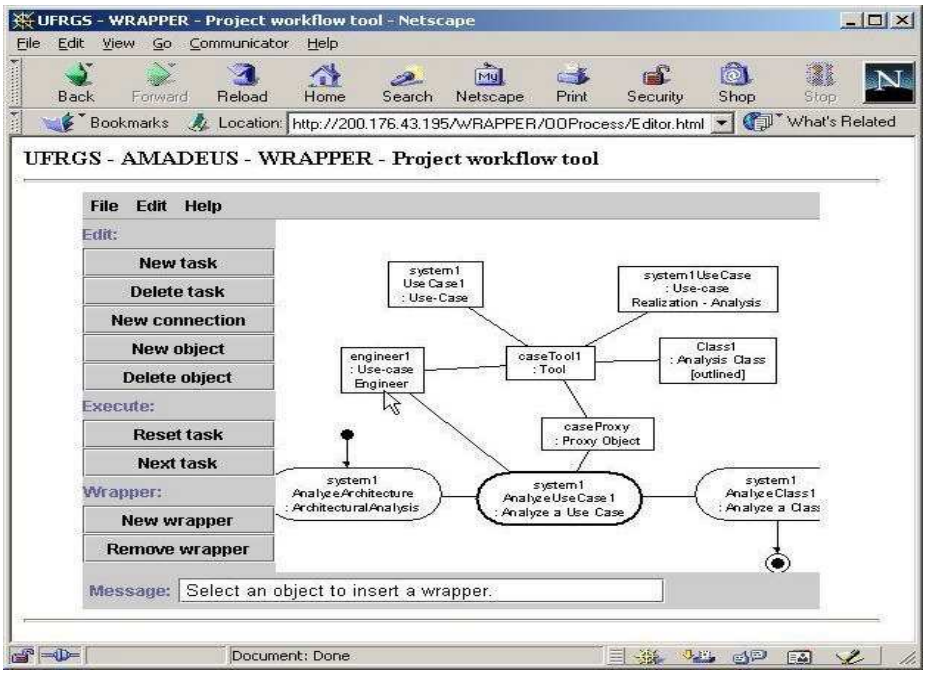

Figura 7: ferramenta de *workflow*.

Os serviços *wrapper* disponíveis são:

- Gravar dados em um arquivo de *log*: toda informação disponível no objeto é armazenada em um arquivo de *log* (que é definido em outra tela).
- Gravar dados em um banco de dados: ao invés de um simples arquivo, os dados coletados por um *wrapper* podem ser armazenados em um banco de dados (uma outra ferramenta é aberta para selecionar os dados do objeto a serem armazenados em um banco de dados específico).
- Ativar um objeto: permite que um outro objeto seja ativado no ORB. Uma tela como a figura 8 é apresentada e o gerente de processo pode indicar o objeto desejado.
- Codificação livre: quando uma ação complexa é desejada é possível codificar as ações desejadas a serem realizadas (o próximo cenário explica esta possibilidade).

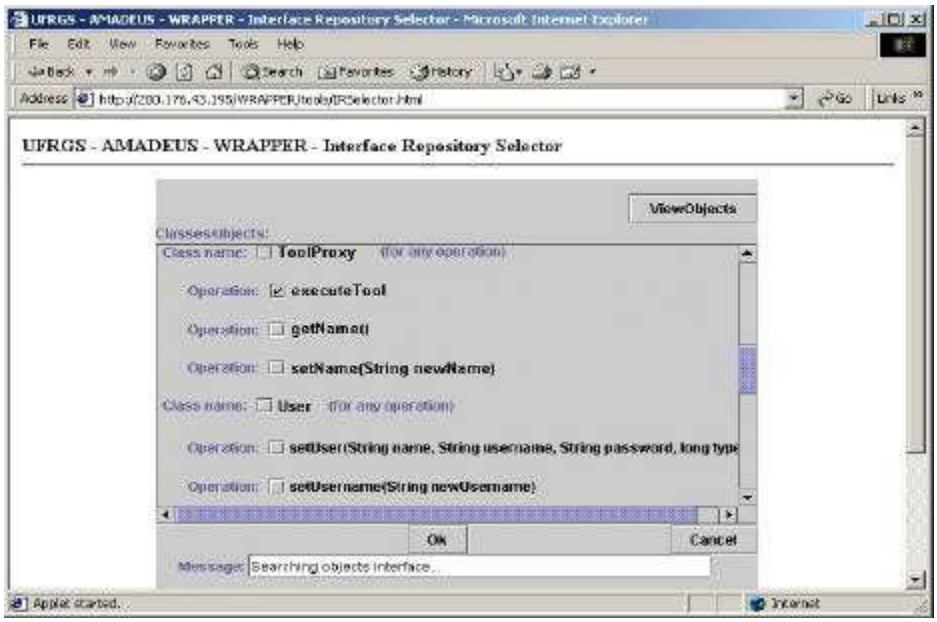

Figura 8: seleção de objeto.

Neste exemplo apresentado, a opção escolhida seria gravar dados em um arquivo de *log*. Este arquivo de *log* ficará disponível para outros programas ou aplicações que podem manipular seu conteúdo (neste exemplo, para realizar análises estatísticas).

6.4.2 Cenário 2: adaptando o processo (distribuição de carga). Neste exemplo, é desejado que se uma ferramenta já estiver em uso e ocorrer uma nova solicitação de execução, o sistema automaticamente chame outra ferramenta.

O registro de ferramentas é realizado por uma tela específica. Para uma ferramenta que não permita o acesso a sua estrutura interna é gerado um objeto-*proxy*, uma vez que a ferramenta é tratada pelo ambiente como um objeto "fechado".

O WfMS ou uma ferramenta separada (figura 8) podem ser usados para selecionar o objeto-*proxy* desejado. Neste exemplo, após a seleção do objeto, o gerente de processo seleciona o Serviço *Wrapper* de codificação livre (figura 9) para codificar uma ação mais complexa. Este serviço ativa uma ferramenta de codificação Java. Adicionalmente a IDL do objeto-*proxy* é apresentada para permitir acesso as definições de sua interface.

O gerente de processo pode então codificar as ações desejadas. O código criado e a definição IDL são usados para gerar o *wrapper* correspondente. Após a geração do *wrapper* correspondente, este é registrado no ORB. A partir de então a tentativa de chamada da ferramenta CASE será interceptada e tratada pelo *wrapper* que pode desviar a chamada para outra ferramenta CASE.

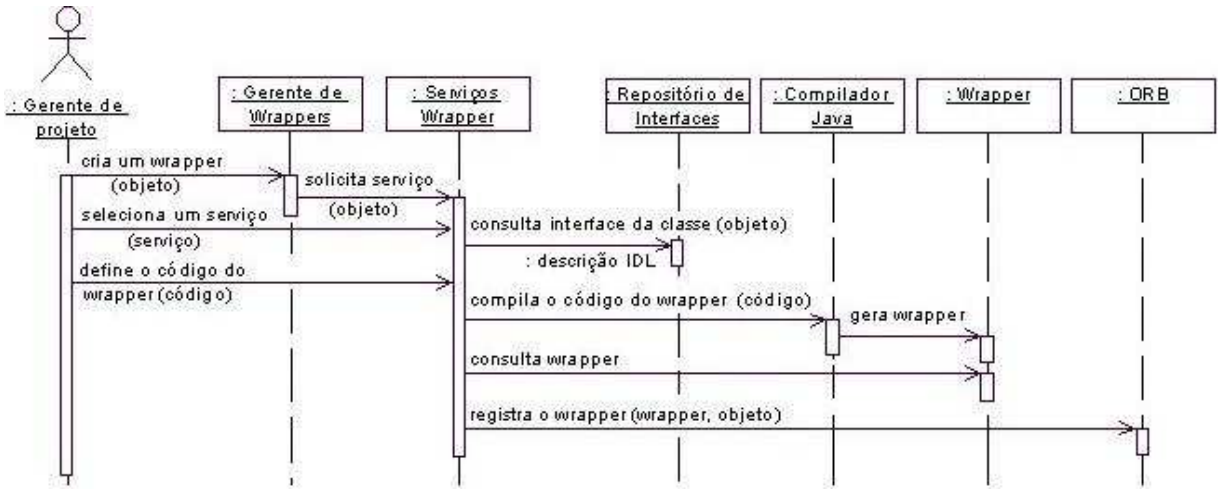

Figura 9: processamento de codificação de *wrapper*.

Os exemplos apresentados mostram que o ambiente permite: o uso de meta-objetos (*wrappers*) para um controle e adaptação do processo de forma flexível; a agregação e uso de ferramentas de plataformas diversas ao ambiente; e o uso do ambiente mesmo em locais remotos e heterogêneos.

6.5 Resultados preliminares

Experimentos de uso do protótipo obtiveram resultados satisfatórios para os requisitos iniciais esperados: flexibilidade, para a gerência de processo pela utilização de meta-objetos; distribuição, para o uso do ambiente em locais remotos pelo uso de tecnologias Web; e heterogeneidade, para o uso do ambiente em plataformas diversas e agregação de ferramentas diversas ao ambiente pelo uso de *middleware* CORBA e tecnologias Web.

Entretanto, uma limitação trazida como o uso de meta-objetos é a sobrecarga na execução do ambiente, com conseqüente perda de desempenho, à medida que novos meta-objetos são inseridos no ambiente. Esta sobrecarga é causada pelos redirecionamentos aos meta-objetos quando uma mensagem entre objetos de nível base é interceptada. Outra

limitação observada é a suscetibilidade do ambiente à infra-estrutura provida pela WWW na execução em clientes remotos, que possui sensível perda de desempenho quando comparado à execução de clientes em rede local.

Considerando-se que o ambiente não visa o desenvolvimento de *software* limitado por requisitos de tempo real a perda de desempenho ainda é aceitável, porém soluções para a melhoria deste aspecto estão sendo projetadas.

7 Trabalhos relacionados

Considerando-se que o presente trabalho utiliza o conhecimento de diversas áreas, abaixo são apresentadas resumidamente os trabalhos semelhantes nas suas áreas específicas:

- Ambientes de Desenvolvimento de *Software* Centrados em Processo: existem diversos PSEEs desenvolvidos em trabalhos similares. SPADE [2], EPOS [19], PROSOFT [14], TABA [20], são alguns exemplos. Cada PSEE apresenta soluções diferentes para os aspectos de modelagem, execução e controle do processo, bem como a distribuição de uso em locais remotos. Apesar de considerar importante o fato de que o processo deve ser reflexivo, não explicitam o uso de reflexão computacional sobre objetos como solução para isto. Adicionalmente, o uso da Web como plataforma de distribuição do ambiente não é explorado em todos os trabalhos.
- Arquiteturas reflexivas: o uso de reflexão computacional para o desenvolvimento de arquiteturas de sistemas também gerou vários trabalhos. Por exemplo: Guaraná [21] (para desenvolvimento de *software* reflexivo em Java), Luthier [6] (para a introspecção de exemplos em Smalltalk), MOTF [15] (para desenvolvimento de aplicações tolerantes a falhas). Entretanto, estas arquiteturas não têm como objetivo o suporte a processo de *software*.
- CORBA reflexivo: o uso de reflexão sobre objetos em CORBA já foi sugerido em alguns trabalhos [7][9][27]. Porém nestes trabalhos a reflexão é utilizada como mecanismo para depuração de aplicações, gerência de objetos distribuídos ou gerência de meta-informação. O uso de reflexão no *middleware* CORBA como mecanismo de controle e adaptação de processo de *software* não é a aplicação principal.

8. Conclusão

Este trabalho apresentou uma arquitetura reflexiva, baseada na Web, para ambiente de suporte a processo de *software*. A arquitetura provê uma infra-estrutura que integra objetos distribuídos e reflexivos, tecnologias Web e *middleware* CORBA.

As principais vantagens trazidas pela arquitetura para um ambiente de desenvolvimento de *software* centrado em processo são:

- flexibilidade: o uso de meta-objetos para o controle de um processo de *software* permite que um gerente de processo possa, a qualquer momento e de maneiras diversas, obter informações sobre a execução do mesmo, bem como alterar o próprio processo. Adicionalmente, o uso de meta-objetos permite que o próprio ambiente seja modificado, uma vez que todos os elementos do ambiente são tratados na forma homogênea de objetos.
- distribuição: o uso de tecnologias Web e objetos distribuídos permite que o uso do ambiente seja realizado em locais remotos. Desta forma, as tarefas envolvidas em um processo podem ser distribuídas remotamente, e os agentes envolvidos em um processo podem realizar estas tarefas em qualquer ponto da *World Wide Web*.
- heterogeneidade: *middleware* CORBA e tecnologias Web permitem que o ambiente possa agregar ferramentas de diversos fornecedores e plataformas ao ambiente.

Além disso, o uso de tecnologias Web permite que o ambiente seja executado em qualquer plataforma cliente.

Os experimentos com o protótipo implementado apresentaram resultados satisfatórios. Com o uso da arquitetura proposta espera-se que o trabalho de um gerente de processo na modelagem, execução e controle de um processo seja facilitado, bem como para que os agentes envolvidos no processo para possam realizar suas tarefas em locais remotos e distribuídos, resultando em uma melhoria na qualidade e produtividade de sistemas de *software*.

8.1. Trabalhos futuros

A estrutura básica da arquitetura já está implementada, entretanto, algumas tarefas adicionais estão previstas:

- novos Serviços *Wrappers* estão em desenvolvimento para facilitar a determinação do comportamento de novos *wrappers* criados no ambiente.
- experimentos com diferentes ORBs CORBA estão sendo realizados para testar a integração de objetos de outros ORBs com o ambiente.
- estratégias alternativas para melhorar o desempenho de execução do processo, como replicação do *workflow* de processo nos clientes e instalação completa do ambiente nos clientes também estão em implementação.

Referências

[1] ARAUJO, R.M.; BORGES, M.R.S. Sobre a aplicabilidade de sistemas de *workflow* no suporte de *software*. In: IDEAS'99 - Workshop iberoamericano de ingeniería de requisitos y ambientes de software, 2. 1999. San José-Costa Rica. **Memorias...** 1999, p. 417-428.

[2] BANDINELLI, S.; FUGETTA, A. Computational reflection in software process modeling: the SLANG approach. In: International conference on software engineering, 15., 1993, Baltimore. **Proceedings...** IEEE Computer Society Press, 1993.

[3] BOOCH, G.; et al. **The unified modeling language user guide**. Boston: Addison-Wesley, 1998.

[4] BORLAND. **Visibroker for Java 4.5: product documentation**. Available in the WWW: http://www.borland.com/techpubs/visibroker/visibroker45/ vbj45-index.html (1°/mai/2002)

[5] BUSCHMANN, F.; et all. **Pattern-oriented software architecture**: a system of patterns. Chichester: John Wiley & Sons, 1996.

[6] CAMPO, M.; PRICE; R.T. Luthier: a framework for building framework visualisation tools. In: FAYAD, M.; JOHNSON, M. **Object-Oriented Application Frameworks**. New York: John Wiley & Sons, 1998.

[7] COSTA, F.M.; BLAIR, G.S. Integrating Meta-Information Management and Reflection in Middleware. In: International symposium on distributed objects & applications, 2., 2000, Antwerp-Belgium. **Proceedings**... 2000, p.133-143.

[8] DERNIAME, J.; et al. **Software Process**: principles, methodology and technology. Lecture Notes in Computer Science, No. 1500. Berlin: Springer, 1999.

[9] DUCHIEN, L.; SEINTURIER, L. Reflective observation of CORBA Applications. In: IASTED International conference on parallel and distributed computing and systems, 11., 1999, Boston-USA. **Proceedings**... IASTED, 1999, p.311-316.

[10] FINKELSTEIN, A.; et al. **Process software modelling and technology**. Somerset: Research Studies Press, 1994.

[11] GIMENES, Itana M. S. Uma introdução ao processo de engenharia de software. In: Jornada de atualização em informática, 13., 1994, Caxambú. **Anais...** SBC, 1994.

[12] HUMPHREY, W.S. **Managing the software process**. Reading: Addison-Wesley: 1990.

[13] JACOBSON, I.; et. al. **The unified software development process**. Reading: Addison-Wesley, 1998.

[14] LIMA REIS, C.; REIS, R.; Nunes, D. Gerenciamento do Processo de Desenvolvimento Cooperativo de Software no Ambiente PROSOFT. Simpósio brasileiro de engenharia de software, 12., 1998, Maringá. **Anais...** SBC, 1998.

[15] LISBOA, M.L.B. **Motf**: meta-objetos para tolerância a falhas. Porto Alegre: UFRGS-PPGC, 1995. (Tese de doutorado)

[16] LISBOA, M.L.B. **Arquiteturas de meta-nível**. In: Simpósio brasileiro de engenharia de software, 11., 1997, Fortaleza. **Tutorial**... Fortaleza: UFC, 1997. 35 p.

[17] MAES, P. Concepts and experiments in computational reflection. In: OOPSLA '89, 1989. **Proceedings...** ACM Sigplan Notices, v.22, n.12, Dec. 1987. p.147-69.

[18] MICROSOFT. **Windows 2000 Server home**. Disponível na WWW em http://www.microsoft.com/windows2000/server/default.asp (1°/5/ 2002)

[19] NGUYEN, M.N.; et al. Total software process model evolution in EPOS. In: International conference on software engineering, 19., 1997, Boston-EUA. **Proceedings...** 1997.

[20] OLIVEIRA, K.M.; et al. **Ambiente de desenvolvimento de software orientado a domínio**. In: Simpósio brasileiro de engenharia de software, 14., 2000, João Pessoa. **Anais...** UFPB: 2000, p. 275-290.

[21] OLIVA, A.; BUZATO, L.E.. The design and implementation of Guaraná. In: USENIX conference on object-oriented technologies and systems, 5., 1999, San Diego-USA. **Proceedings...** 1999.

[22] OBJECT MANAGEMENT GROUP. **Complete formal CORBA/IIOP 2.3 specification**. Disponível por WWW: http://www.omg.org/cgi-bin/doc?formal/98-12-01 (1º/mai/2002)

[23] OBJECT MANAGEMENT GROUP. **Interceptors**. Disponível na WWW em http://www.omg.org/cgi-bin/doc?formal/99-07-25 (1º/mai/ 2002)

[24] PRESSMAN, R.S. **Software engineering**: a practitioner's approach. New York: McGraw-Hill, 2001.

[25] SHARON, D.; BELL, R. Tools that bind: creating integrated environments. **IEEE Software**, Los Alamitos, v.12, n.2, p.76-85, Mar. 1995.

[26] SUN MICROSYSTEMS. **The Java tutorial**. Available in the WWW: http://java.sun.com/docs/books/tutorial (1°/mai/2002)

[27] WEGDAN, M.; et al. **Using Message Reflection in a Management Architecture for CORBA. In:** IFIP/IEEE International workshop on distributed systems: operations & management, 11, 2000. Austin-USA. **Proceedings...** IEEE: 2000.

[28] WORKFLOW MANAGEMENT COALITION. **Interface 1 - process definition interchange process model**. Available in the WWW: www.wfmc.org/standards/docs/TC-1016-P_v11_IF1_Process_definition_ Interchange.pdf (1°/mai/2002)