LuaSpace EPlus – Um Ambiente para Desenvolvimento de Aplicações CORBA no Eclipse

André Gustavo Duarte de Almeida, Thais Batista, Nélio Cacho

Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário – Lagoa Nova – 59.072-970 - Natal - RN

andre@consiste.dimap.ufrn.br, thais@ufrnet.br,
cacho@consiste.dimap.ufrn.br

Resumo. *Esse artigo apresenta um ambiente para desenvolvimento interativo de aplicações CORBA disponível no Eclipse - LuaSpace EPlus. Esse ambiente dispõe de várias ferramentas para facilitar o processo de desenvolvimento, promover o reuso de componentes e oferecer suporte à reconfiguração dinâmica das aplicações. As ferramentas que compõem o ambiente são: (1) uma interface gráfica com editores, navegadores para repositórios CORBA e assistentes que incluem geração automática de código das aplicações; (2) um mecanismo de seleção dinâmica de componentes associado com um serviço para definição, publicação e consulta de ontologias; (3) uma linguagem de programação baseada em aspectos; (4) uma biblioteca que facilita o uso do serviço de segurança de CORBA e inclui funcionalidades adicionais para segurança.*

Abstract. *In this paper we present a CORBA-based interactive development environment available in Eclipse – LuaSpace EPlus. This environment provides a set of tools to make the development process easier, to promote components reuse and to support dynamic reconfiguration. The tools that compose this environment are: (1) a graphical interface with editors, browsers to CORBA repositories and wizards that include automatic generation of application code; (2) a mechanism for the dynamic selection of components that is combined with a tool to the definition, publication and query of ontologies; (3) an aspect-based programming language; (4) a library that makes the use of the CORBA Security Service easier and includes additional functionalities.*

1. Introdução

Plataformas de middleware [Bernstein 1996] oferecem uma infra-estrutura para facilitar o desenvolvimento baseado em componentes definindo formas padrão para declaração de interfaces de componentes e para comunicação entre os componentes distribuídos. CORBA (*Common Object Broker Request Architecture*) [OMG 2004] é uma plataforma de *middleware* que tem se destacado em relação às demais por ser uma especificação aberta, independente de fabricante e de linguagem.

Para incentivar o desenvolvimento de aplicações CORBA através do reuso de componentes é necessário prover um ambiente de desenvolvimento que ofereça facilidades para programação da aplicação e para encontrar componentes disponíveis para o reuso. LuaSpace [Batista and Rodriguez, 2000] é um ambiente para

desenvolvimento de aplicações CORBA que usa uma linguagem de configuração para definição da estrutura de aplicações formadas por componentes CORBA e para dar suporte à reconfiguração dinâmica da aplicação. O ambiente explora uma linguagem interpretada e dinamicamente tipada – a linguagem Lua [Ierusalimschy 1996] – e um *binding* dinâmico entre Lua e CORBA – LuaOrb [Cerqueira 1999] – que possibilita o acesso dinâmico a componentes CORBA da mesma maneira que se faz acesso a objetos Lua. LuaSpace Plus [Almeida et al 2004] é uma versão do LuaSpace que oferece uma interface grafica composta por um editor de textos e um console embutido para execução das aplicações. Além disso, provê um mecanismo para seleção dinâmica e uma biblioteca para segurança de aplicações. Apesar dessas facilidades serem importantes ferramentas auxiliares no desenvolvimento de aplicações, elas não são suficientes para proporcionar um poderoso ambiente de desenvolvimento de fácil instalação e uso. Um dos principais entraves do uso de ambientes de desenvolvimento baseados em CORBA está na complexidade de instalação e configuração do ambiente.

Outro importante obstáculo é a complexidade de uso dos serviços CORBA. Portanto, para facilitar o desenvolvimento, o ambiente deve prover um conjunto de ferramentas baseadas em CORBA que forneçam ao programador uma abstração em relação às complexidades de uso de serviços CORBA. Seleção dinâmica e segurança são serviços comumente usados por uma variada gama de aplicações. Seleção dinâmica é crucial para o reuso de componentes. Segurança é um serviço essencial em um ambiente distribuído.

Em termos de seleção dinâmica, CORBA oferece os serviços de Nomes e de *Trading*. No entanto, esses serviços não são suficientemente expressivos para aceitar vários critérios para seleção de componentes e, inclusive, a combinação de vários critérios em uma determinada seleção. Outro problema de tais serviços que representa um obstáculo para a busca de componentes é a ausência de uma terminologia comum para evitar problemas semânticos. Portanto, há necessidade de aliar, um serviço de seleção dinâmica mais expressivo com um sistema de ontologias que estabeleça terminologias padrão a serem seguidas na definição e publicação dos componentes bem como na descoberta de componentes. O resultado da seleção dinâmica deve ser não apenas as referências dos objetos que satisfazem os critérios, mas principalmente, as interfaces dos componentes selecionados pelo serviço.

Em relação à segurança, o serviço de segurança de CORBA, apesar de poderoso, é bastante complexo de utilizar, o que dificulta o desenvolvimento de aplicações que necessitam de mecanismos de segurança.

Ambientes para desenvolvimento baseado em componentes (DBC) focam na separação de conceitos através da dissociação entre interface e implementação de forma a promover o reuso. A programação orientada a aspectos (POA) [Elrad et al. 2001] compartilha com o DBC a idéia de separação de conceitos para promover reuso através da dissociação entre o código essencial dos componentes e o código que não faz parte da funcionalidade básica do componente, mas que, tradicionalmente, está espalhado em diversos componentes de uma aplicação. A POA oferece uma estratégia para separar esse código, modularizando-o em um elemento chamado *Aspecto*. Tal estratégia de separação entre o código do componente e o código do aspecto favorece o reuso de componentes e diminui a complexidade de desenvolvimento. O mesmo componente pode ser usado em vários contextos combinados com diferentes aspectos. Um

mecanismo de composição (*weaving*) realiza a combinação do código do aspecto com o do componente alvo do aspecto.

Esse trabalho apresenta o ambiente LuaSpace EPlus que estende as versões anteriores do LuaSpace de forma a tratar os problemas mencionados acima. Visando proporcionar um poderoso ambiente de desenvolvimento de fácil instalação e uso, o LuaSpace EPlus foi desenvolvido usando o Eclipse [Eclipse] que se constitui em uma poderosa ferramenta de desenvolvimento de software, sendo um dos ambientes de desenvolvimento integrado (IDE) mais utilizados no mundo e que oferece uma gama de recursos para facilitar e agilizar o processo de desenvolvimento. A disponibilidade do ambiente LuaSpace no Eclipse vem promover a divulgação e o uso desse ambiente, e eliminar a complexidade de instalação e configuração. O Eclipse provê mecanismos de extensão que permitem modificar o seu ambiente de desenvolvimento para adaptá-lo a requisitos de outros ambientes que sejam instalados nessa IDE. Os mecanismos de extensão foram usados na implementação do LuaSpace EPlus.

Além da interface gráfica, incluindo editores, *browsers* e assistentes associados com geradores de código, o LuaSpace Eplus integra várias ferramentas que foram desenvolvidas em separado: (1) um serviço CORBA para seleção dinâmica de componentes – o Discovery Service [Cacho et al. 2004] - que está associado com um serviço para publicação e consulta de ontologias – o OntoService; (2) uma biblioteca com funções de segurança – o LOrbSec [Martins et al. 2004]; (3) uma linguagem de programação orientada a aspectos – AspectLua [Cacho et al. 2005]- baseada na linguagem Lua.

Este artigo está organizado da seguinte forma. A seção 2 apresenta, resumidamente, os conceitos básicos relacionados a esse trabalho e as ferramentas para seleção dinâmica associada com ontologia, segurança e programação orientada a aspectos. A seção 3 apresenta o ambiente gráfico interativo para desenvolvimento de aplicações CORBA e as ferramentas desse ambiente. A seção 4 ilustra um estudo de caso que explora parte das funcionalidades do ambiente. A seção 5 comenta sobre trabalhos relacionados. A seção 6 contém as conclusões.

2. Conceitos Básicos

2.1. LuaSpace Plus

O ambiente LuaSpace combina a plataforma CORBA com a linguagem Lua permitindo que uma aplicação escrita em Lua possa ser composta por componentes implementados em qualquer linguagem que tenha o *binding* para CORBA. O ambiente é composto pela linguagem Lua e por LuaOrb – um *binding* entre Lua e CORBA baseado nas interfaces dinâmicas de CORBA – Interface de Invocação Dinâmica (DII) e Interface de Esqueleto Dinâmico (DSI).

Lua é uma linguagem interpretada com um sistema de tipos dinâmico: variáveis não têm tipo, apenas valores estão associados a um tipo. A linguagem possui algumas características não convencionais: funções são valores de primeira classe; *arrays associativos* (chamados *tabelas*) são a única facilidade de estruturação de dados; *metatables* é o mecanismo genérico para reflexão que são usados para serem chamados em situações onde o interpretador Lua não tem como proceder. Através desse mecanismo as invocações para objetos CORBA são direcionadas para LuaOrb tratar. Para usar um componente CORBA é necessário primeiro criar um *proxy* Lua que

representa o objeto CORBA. As operações aplicadas sobre esse *proxy* são tratadas por LuaOrb que as transforma em operações sobre componentes CORBA.

2.2. AspectLua

AspectLua [Cacho et al. 2005] é uma extensão da linguagem Lua para POA. Essa extensão explora as facilidades reflexivas de Lua e o conceito de tabelas na definição dos elementos que compõem o aspecto: os pontos de junção aonde o aspecto irá atuar (*join points*) e a ação a ser executada pelo aspecto (*advice*) ao encontrar o ponto de junção. O processo de associação do aspecto com o componente que contém o ponto de junção é realizado por AspectLua em tempo de execução.

Simplicidade e flexibilidade são as principais características da linguagem Lua. AspectLua segue a mesma idéia e preserva tais características. Apenas duas instruções são necessárias para definição de aspectos, pontos de junção e *advice*. A criação do aspecto é feita através da definição de uma tabela Lua cujos elementos compõem o aspecto. AspectLua define um objeto *Aspect* que oferece a função *new()* para criar o aspecto. Depois de criar o aspecto é necessário definir a tabela Lua que contem o nome do aspecto, os pontos de junção e o *advice*. A Figura 1 ilustra a criação de um aspecto. A linha 3 apresenta a definição do aspecto com os seguintes parâmetros: (1) o nome do aspecto é *aspectLog*; (2) o nome do ponto de junção é *logpoint* que irá interceptar chamadas (*Call*) ao método *print*. (3) a função *printLog* será invocada antes (*before*) do ponto de junção.

```
1 require “AspectLua.lua”
2 a = Aspect:new
3 a:aspect({name = 'aspectLog'},
            {name = 'logpoint',designator = 'Call',list =  {'print'}},
            {type = 'Before', action = printLog})
```

Figura 1. Criação de um Aspecto

2.3. LOrbSec

LOrbSec [Martins et al 2004] é uma biblioteca que consiste em uma abstração sobre o serviço de segurança de CORBA (CORBASec) e oferece funcionalidades adicionais não disponíveis no CORBASec: verificação de certificados e associação de nomes de domínios a principais. LOrbSec oferece funções para autenticação, controle de acesso, auditoria e verificação de certificados. A Tabela 1 contém a lista das funções utilizadas para autenticação e controle de acesso.

Para gerenciar o processo de autenticação o LOrbSec disponibiliza as funções *create_certificate* e *authenticate*. A função *create_certificate* recebe como parâmetro as informações do proprietário (nome, e-mail, cidade, empresa, setor, etc.) e retorna uma referência para um objeto que será usado para chamar os métodos *get_certificate* e *get_key*. Estes dois métodos permitem salvar, em arquivo, os valores dos certificados e das chaves públicas. Com os valores do certificado e da chave pública em arquivo é possível invocar o método *authenticate*. Este método recebe como parâmetro o par certificado/chave e fornece a referência do objeto que será usado para chamar o método *authentication_state* que retorna o status da autenticação. Portanto, para realizar o processo de autenticação pode-se usar a chamada *status* =

LorbSecLevel2:authenticate("ServCert.pem","ServKey.pem") e para obter o status da autenticação pode-se usar *status:authentication_state()*.

Tabela 1. LOrbSec - Funções

Funcionalidade	Métodos	Descrição
Autenticação	create_certificate()	Cria certificados
	Get_certificate()	Obtém certificados
	Get_key()	Obtém a chave do certificado
	authenticate()	Realiza a autenticação
	Authentication_state	Obtém estado da autenticação
Controle de Acesso	create_domain()	Define a hierarquia de domínios
	set_required_rights()	Define os direitos requeridos
	grant_rights()	Define os direitos garantidos

O controle de acesso de uma aplicação depende, inicialmente, da construção da hierarquia de domínios, seguida pela distribuição, nos devidos domínios, dos objetos que possuem restrição de acesso e, por fim, a atribuição dos direitos aos usuários. A hierarquia de domínios permite agrupar, em cada domínio, as operações que serão realizadas por um determinado grupo de usuários. Na biblioteca LOrbSec domínios são criados através do método *create_domain* que recebe como parâmetro uma *string* contendo o nome do domínio. O nome do domínio é representado de forma semelhante ao sistema de arquivo UNIX, onde o domínio raiz (*root*) é sempre (/) e chamadas como *create_domain("/Dimap/Professores")* criam o domínio *Professores* que é sub domínio de *Dimap*. Após a construção da hierarquia de domínios é necessário definir quais as operações que irão requerer direitos para sua execução e associá-las a seus devidos domínios. Para isso deve-se usar o método *set_required_rights*. Este método recebe como parâmetro o nome do domínio, o nome da interface, o nome da operação, um Combinador e os direitos requisitados.

2.4. *Discovery Service* e *OntoService*

O serviço de descoberta (*Discovery Service*) [Cacho et al 2004] apresenta uma maneira simples e uniforme de se buscar componentes considerando diferentes critérios de busca: nome de métodos, assinatura de métodos, exceções e propriedades não funcionais. Esse serviço permite buscas síncronas e assíncronas. Buscas assíncronas são tratadas através de um mecanismo de *callback* que notifica o cliente do serviço quando acontece um registro de um novo componente que satisfaz um critério previamente estabelecido pelo cliente. As principais funções oferecidas pelo serviço são: *insertService*, *search,* e *search_back.*

Para facilitar a publicação e busca de informações por diferentes usuários é necessário haver uma padronização de termos referente a um dado domínio que serve como referência para publicação de componentes desse domínio. Ontologias têm sido utilizadas para esse fim visando fornecer um consenso de terminologias para evitar problemas semânticos. O *OntoService* é um serviço que oferece uma interface para publicação, consulta, exportação e importação de ontologias A combinação desse serviço com o serviço de descoberta confere mais poder de expressão aos critérios de seleção e evita problemas semânticos que inviabilizam uma busca.

O OntoService provê uma interface para gerenciar os modelos ontológicos. Os seguintes métodos são oferecidos: *createModel, removeModel, getModel* e *getAllModels*. A criação de um modelo permite a definição de uma ontologia para determinado domínio. O OntoService possibilita a definição das classes que compõem um modelo e as propriedades de cada classe. O serviço também permite a definição de relações entre classes e entre propriedades. Pode-se, por exemplo, especificar que duas ou mais propriedades são equivalentes. O OntoService oferece uma linguagem de consulta, baseada em SQL, para se especificar uma consulta a uma ontologia.

A integração do *Discovery* com o OntoService trabalha da seguinte forma. O processo de registro de um componente, no *Discovery*, utiliza os modelos ontológicos previamente estabelecidos. No momento do registro de um componente, o *Discovery* consulta o modelo ontológico que o componente a ser registrado está seguindo e verifica se as propriedades exigidas no modelo estão sendo especificadas. Da mesma forma, para uma consulta ao *Discovery*, o modelo ontológico de um determinado domínio deve ser consultado pelo usuário antes de estabelecer os critérios de busca. Isso evita que o usuário utilize termos diferentes dos usados no domínio, que não tenham equivalência estabelecida no modelo ontológico e que o processo de busca não irá ter sucesso. Após conhecer a terminologia usada no domínio o usuário pode estabelecer os critérios da busca que será efetivada pelo *Discovery*. Nesse processo de busca, o *Discovery* consulta os modelos ontológicos definidos pelo usuário para verificar a equivalência entre propriedades e classes.

Na incorporação do OntoService com o *Discovery*, a função *search* foi estendida para permitir que um dos seus parâmetros seja o modelo ontológico que a busca deve seguir. Por exemplo, na invocação *search("(ppm > 5) and (hasColor == 'Black')", "Office")* o *Discovery* e o *Ontoservice* são combinados parar retornar todas as instâncias de objetos que possuam propriedades compatíveis com as solicitadas (*ppm* e *hasColor*) no modelo ontológico *Office*. O *Discovery* pode ampliar a busca consultando propriedades equivalentes a *ppm* ou *hasColor* que possam também estar definidas, como *speed* ou *printcolor.* Esta funcionalidade incrementa o alcance da busca e permite uma maior reutilização dos objetos registrados no *Discovery*.

3. LuaSpace EPlus

3.1. Arquitetura

A Figura 2 ilustra a arquitetura do ambiente LuaSpace EPlus incluindo a interface gráfica e a arquitetura subjacente. O LuaSpace EPlus consiste no ambiente LuaSpace com extensões que incluem: (1) a biblioteca de segurança LOrbSec; (2) um serviço CORBA para seleção dinâmica de componentes, o *Discovery Service* [Cacho, 2004] que, associado com o *OntoService*, constitui uma poderosa ferramenta para seleção dinâmica sua integração com os componentes; (3) o AspectLua que permite a definição de aspectos e sua integração com os componentes e (5) um *plugin* para a IDE Eclipse que provê mecanismos para utilização das ferramentas que compõe o LuaSpace de forma simples e intuitiva e inclui janelas para edição de código, assistentes para geração de código, navegador de aspectos integrado com o editor mostrando os pontos de interceptação do aspecto, navegador de interfaces, configurador de preferências além dos demais utilitários normalmente fornecidos pelo Eclipse para o desenvolvimento de aplicações.

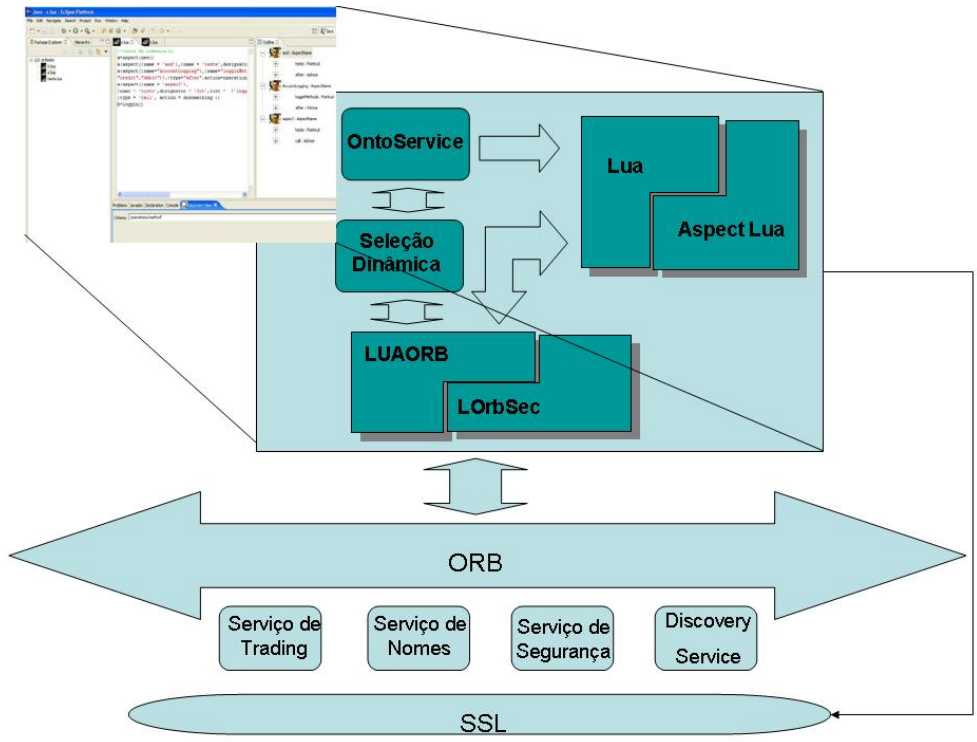

Figura 2. O Ambiente LuaSpace EPlus

A extensão do ambiente em relação à seleção dinâmica e segurança explora serviços CORBA de forma a evitar soluções proprietárias. O LuaSpace EPlus habilita automaticamente os serviços tornado-os operacionais para uso a qualquer momento.

No LuaSpace o script de configuração da aplicação é submetido ao interpretador Lua que o executa fazendo chamadas a cada diferente componente da arquitetura para tratar os comandos.

LuaOrb é acionado nos casos de chamadas que fazem acesso a objetos CORBA. LuaOrb faz o mapeamento entre a chamada Lua e a chamada correspondente na plataforma CORBA. LuaOrb também é invocado quando são feitas chamadas para instalação dinâmica de objetos Lua em um servidor remoto.

Quando o interpretador Lua executa uma chamada de funções de segurança, ele invoca a implementação da função disponível na biblioteca LOrbSec. LOrbSec possui também autonomia de comunicar-se diretamente com outras bibliotecas que não fazem parte do ambiente LuaSpace, como a API SSL, para dar suporte a funcionalidades não fornecidas pelo CORBASec.

O *Discovery Service* é invocado quando a aplicação publica ou consulta componentes. Os parâmetros das funções de seleção dinâmica seguem as definições de conceitos presentes no *OntoService*. O Discovery interage com o OntoService para verificar se os componentes registrados seguem corretamente as definições ontológicas. O Discovery auxilia no processo de busca de componentes através da substituição de termos equivalentes.

AspectLua é invocado quando a aplicação utiliza aspectos. Como AspectLua é uma extensão da própria linguagem Lua, a utilização em conjunto com as demais ferramentas é realizada de forma natural, não sendo necessário nenhum adaptação das demais ferramentas.

3.2. Interface Gráfica e Ferramentas

A Figura 3 ilustra a interface gráfica do ambiente. As duas primeiras janelas são os *browsers* de projeto e de interfaces de componentes, respectivamente. A terceira janela é o editor de programas. A quarta janela contém o *browser* de aspectos. Na parte inferior há uma quinta janela que corresponde ao console de execução.

Além das janelas, a interface oferece diversos assistentes. Os assistentes são poderosas ferramentas no desenvolvimento de software, visto que economizam tempo e conferem rapidez e simplicidade ao desenvolvimento além de evitar erros. As ferramentas do LuaSpace são utilizadas nas aplicações através de código Lua, o que obriga o programador a digitar várias linhas de código. Esse passo não é necessário com a utilização dos assistentes. LuaSpace inclui os seguintes assistentes: (1) assistente para definição de aspectos, (2) assistente para definição das prerrogativas de segurança, (3) assistente para definição de ontologias utilizando o *OntoService,* (4) assistente para criação de projetos LuaSpace e arquivos Lua, (5) assistente para publicação de interfaces (6) assistente para utilização (consulta e publicação) do *Discovery Service*.

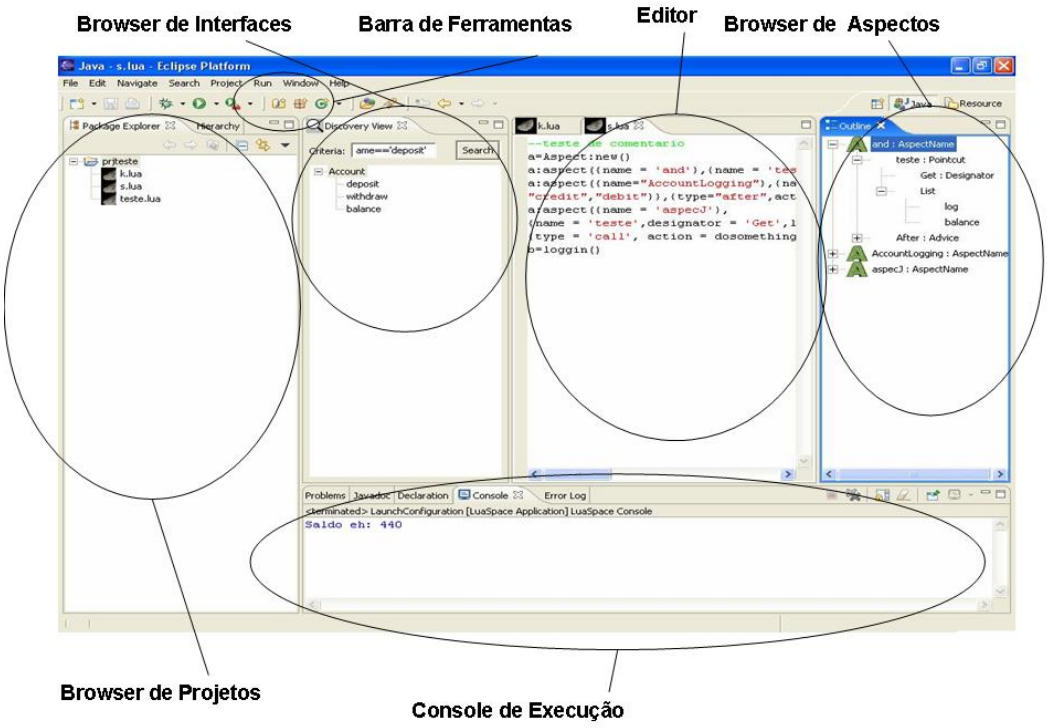

Figura 3. Interface gráfica do ambiente de desenvolvimento LuaSpace Eplus

A Figura 4 ilustra o assistente para criação de aspectos. Nesse assistente o programador especifica os elementos que constituem um aspecto segundo as definições do AspectLua. Se os dados forem colocados de forma correta, ao pressionar o botão *Finish*, o código do Aspecto é gerado automaticamente.

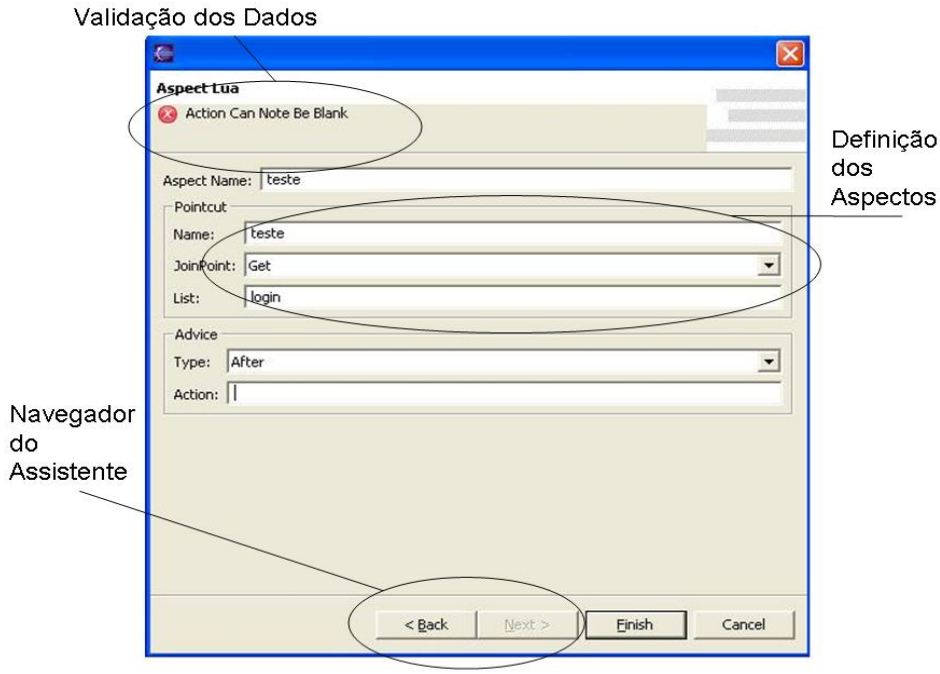

Figura 4. Assistente de criação de aspectos

O assistente para definição de funções relacionadas com segurança é composto por 4 páginas que se referem aos 4 elementos de segurança que podem ser definidos através do LOrbSec. As páginas *DomainPage, CertificatePage, AccessControlPage* e *AuditControlPage* são responsáveis respectivamente pela criação dos domínios, criação de certificados, controle de acesso e auditoria. Na Figura 5 está à página responsável pela criação dos elementos relacionados aos certificados do tipo X.509.

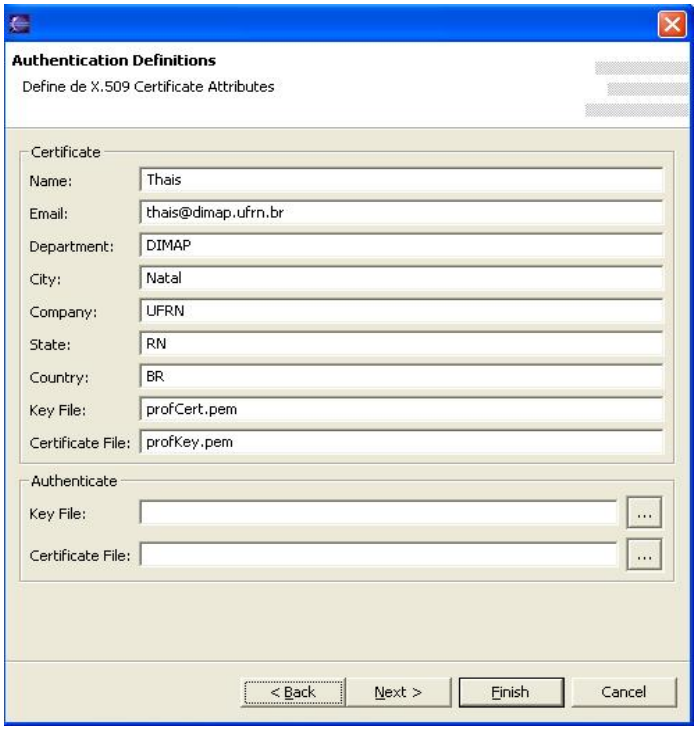

Figura 5. Assistente para definição de Certificados

4. Estudo de Caso

Nesta seção apresentamos um estudo de caso para demonstrar, via uma aplicação exemplo, parte das funcionalidades que compõe o LuaSpace EPlus. A aplicação é composta por vários servidores de impressão que disponibilizam a operação *print* para os clientes que devem ser autenticados para utilizar tal operação. A Figura 6 descreve a interface do servidor de impressão. A interface disponibiliza os métodos *print* e *configure* que servem, respectivamente, para imprimir, a partir de um arquivo fornecido como parâmetro, e configurar o tipo de papel utilizado.

```
1 interface Account{
2         void print(in string filename);
3         void configure(int string papertype);
4 };
```

Figura 6. Interface *Account*

Através do *browser* de interfaces é possível publicar uma interface no servidor alem de definir as propriedades do objeto como ilustrado na Figura 7.

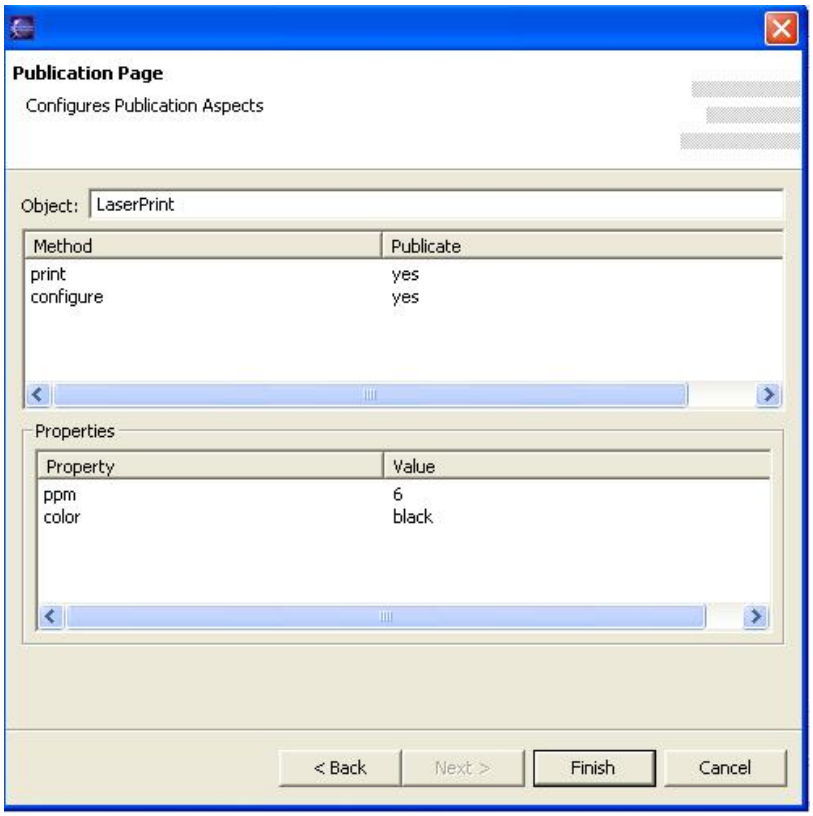

Figura 7. Publicação da Interface *LaserPrint*

Através do assistente de publicação o usuário pode definir quais métodos da interface devem ser publicados e quais as propriedades a serem adicionadas ao serviço de descoberta. Na Figura 7 a interface *LaserPrint* terá os métodos *print* e *configure* publicados. O assistente gera o código relativo à implementação do serviço como ilustra a Figura 8, das linhas 8 a 15. Em seguida, o usuário define as propriedades que deverão

ser associadas à implementação, gerando o trecho de código correspondente (ilustrado nas linhas 16 e 17 da Figura 8).

Em seguida são definidas as prerrogativas de segurança. No caso da aplicação em questão, o usuário deve possuir um certificado que o identifique garantindo assim confiabilidade da invocação. A Figura 5 contém os dados necessários.

Na Figura 8 há o trecho de código, gerado automaticamente pelo gerador de códigos do ambiente, a partir das prerrogativas definidas na Figura 5.

```
1 Cert = SecurityLeve2:create_certificate("BR","BR","RN","Natal","UFRN","DIMAP","Thais",
                                          "thais@dimap.ufrn.br")
2 writeto("profCert.pem")
3 write(cert:get_cert()
4 writeto()
5 writeto("profKey.pem")
6 write(cert:get_key()
7 writeto()
8 LaserPrint={
9         print=function(self,filename)
10        io.write(filename)
11        end,
12        configure=function(self,papertype)
13        io.write("PapertType: "..papertype)
14        end
15  }
16 properties={ppm=6,color="black"}
17 servant=luaorb.createsrvant(LaserPrint,"IDL:LaserPrint",properties)
```

Figura 8. Código gerado após as configurações realizadas

Na linha 1 há a chamada da função *create_certificate*, da biblioteca LOrbSec, que é responsável por criar certificados do tipo X.509. Nas linhas 2 a 7 as referências ao certificado e a chave são guardadas no arquivo para recuperação posterior. As linhas 8 a 15 descrevem a implementação do serviço de impressão. Na linha 16 é construída uma tabela Lua com as propriedades desse serviço e na linha 17 é registrada a implementação do serviço.

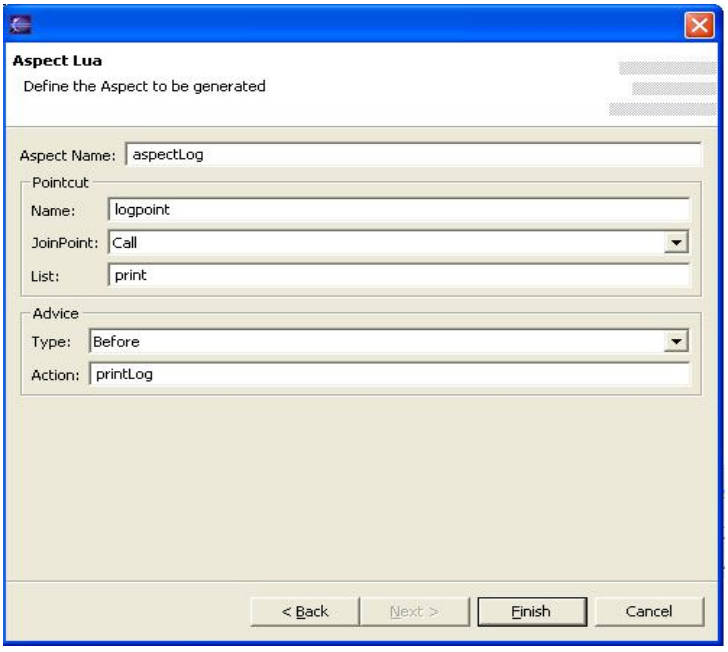

Figura 9. Definição dos aspectos da aplicação

Como mencionado anteriormente, um dos requisitos da aplicação é que seja registrado, em um arquivo de log, todas as operações de impressão com o intuito de fornecer um histórico das horas em que o serviço é mais utilizado. Todas as chamadas ao método *print* devem ser catalogadas. A Figura 9 ilustra a utilização do assistente de criação de aspectos.

Na Figura 10 encontra-se o código gerado pelo assistente de criação de aspectos. As linhas 1 a 3 contêm a implementação da função responsável por registrar o log das operações. Na linha 4, através da chamada *Aspect:new()*, a classe *Aspect* é instanciada. A linha 5 contém a chamada para criação do aspecto que informa que, em toda chamada do método *print*, a função *printLog* deve ser invocada para registrar a utilização do serviço de impressão.

```
1 function printLog()
2           io.write("Print Operation Requested at: "..os.date())
3 end
4 a=Aspect:new()
5 a:aspect({name = 'aspectLog'},
            {name = 'logpoint',designator = 'Call',list =  {'print'}},
            {type = 'Before', action = printLog})
```

Figura 10. Código para definição dos Aspectos

Para realizar a integração entre o *Discovery* e o *OntoService* é necessário criar um modelo ontológico que mostre as relações entre as classes que compõe o serviço.

Através do assistente para criação de ontologias é possível criar classes e propriedades e definir as relações entre as classes para que o sistema de busca possa realizar as substituições necessárias. A Figura 11 mostra os passos para criação do modelo utilizando o assistente provido pelo ambiente. São adicionadas 3 classes ao modelo: (1) Printer que representa uma impressora genérica;(2) InkJet que representa uma impressora jato de tinta; (3) LaserPrint que representa uma impressora laser. As propriedades *hasSpeed* e *ppm* indicam a velocidade de impressão.

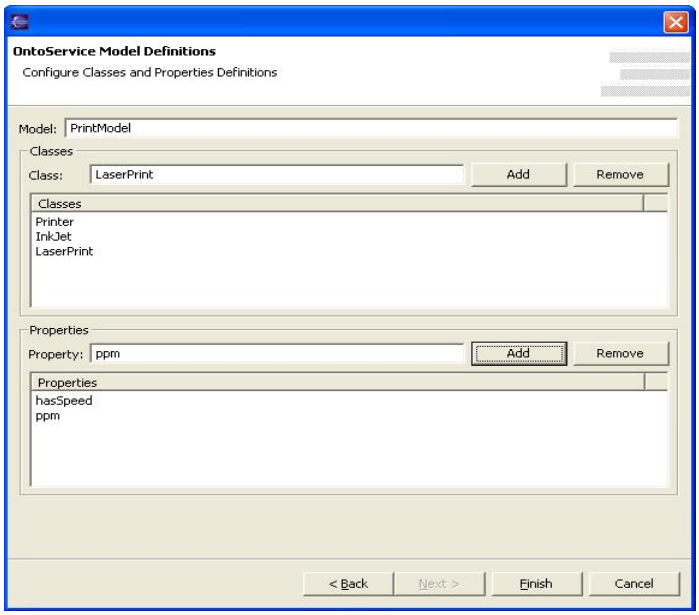

Figura 11. Assistente para criação do modelo ontológico

A Figura 12 mostra o código gerado após a execução do assistente de criação do modelo ontológico.

```
1  model=OntoService:createModel("PrintModel")
2  Printer=model:createClass("Printer")
3  InkJet=model:createClass("InkJet")
4  LaserPrint=model:createClass("LaserPrint")
5  hasSpeed=model:createProperty("hasSpeed")
6  ppm=model:createProperty("ppm")
7  InkJet:addSubClassOf(Printer)
8  LaserPrint:addSubClassOf(Printer)
9  hasSpeed:addDomain(Printer)
10 hasSpeed:addEquivalentProperty(ppm)
11 ppm:addEquivalentProperty(hasSpeed)
```

Figura 12. Código gerado para criação do modelo ontológico

Na linha 1 é criado o modelo através da chamada do serviço OntoService. Nas linhas 2 a 4 são criadas as classes que compõem o modelo. Nas linhas 5 e 6 são criadas as propriedades. As linhas 7 e 8 informam que as classes *InkJet* e *LaserPrint* são subclasses da classe *Printer*. Na linha 9 a propriedade *hasSpeed* é associada à classe *Printer*. As linhas 10 e 11 definem que as propriedades *hasSpeed* e *ppm* são equivalentes.

```
1 Status=SecurityLevel2:authenticate(“profCert.pem”,”profKey.pem”)
2 If (Status:authenticate_state()==”Sucess”) then
3          Cli=Generic()
4          Cli:print(“arquivo.ps”)(“hasSpeed<10”)
5 Else
6          Io.write(“Failure”)
7 end
```

Figura 13. Código que utiliza a implementação do serviço de impressão

Para utilizar o serviço de impressão o usuário deve apresentar um certificado e a respectiva chave para autenticação. A liberação do uso acontece em caso de sucesso na autenticação. No trecho de código da Figura 13 é ilustrado o processo de autenticação de certificados pelo LOrbSec. Caso o usuário seja autenticado com sucesso é realizada uma busca, no Discovery, por um objeto qualquer que possua a propriedade *hasSpeed < 10*. No momento em que foi publicado o serviço informava a propriedade *ppm*, porém, ao realizar a busca o OntoService é consultado, o que resulta na substituição da propriedade *hasSpeed* no critério de busca por ppm localizando com sucesso o objeto.

A Figura 14 ilustra o *browser* de interfaces que é disponibilizado para o usuário navegar no repositório de interfaces, em tempo de desenvolvimento, permitindo que o mesmo tenha conhecimento das interfaces e objetos que satisfazem os critérios de seleção por ele especificado. O *browser* possui espaço para especificação dos critérios de seleção (lado esquerdo), e logo abaixo desse espaço, o *browser* apresenta uma estrutura em forma de árvore de todas as interfaces e objetos que satisfazem o critério de seleção especificado. A janela do lado direito representa o editor de textos do ambiente, independente do *browser* de interfaces, contendo um código de um modelo ontológico.

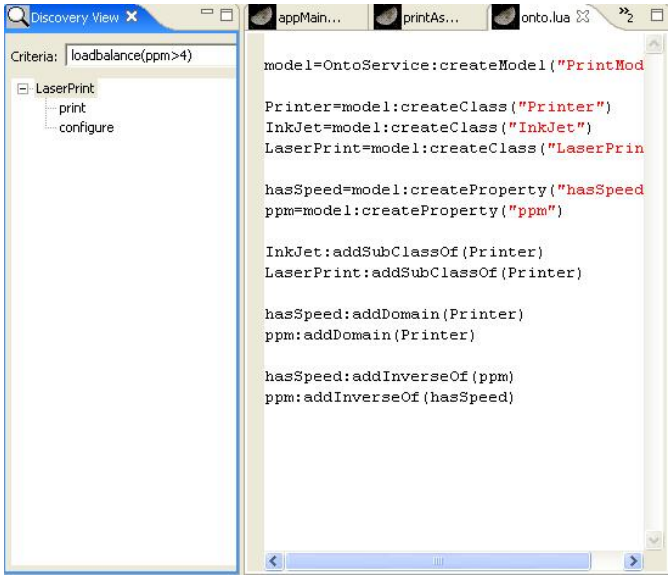

Figura 14. Utilização do *Discovery* via interface

5. Trabalhos Relacionados

Os ambientes *Áster* e *Olan* foram as primeiras iniciativas em termos de ambiente de desenvolvimento de software baseado em componentes compartilhando com o LuaSpace EPlus a necessidade de facilitar o desenvolvimento de aplicações baseadas em componentes através de um ambiente gráfico. O *Olan* segue um modelo de componentes proprietário e dispõe de uma ferramenta gráfica onde é possível construir modelos e realizar a interconexão entre eles. Nenhum desses ambientes inclui ferramenta adicional de suporte a seleção dinâmica, ontologias e segurança nem suporte a POA.

O *Fractal* [Bruneton et al 2002] é um modelo de componentes que usa programação orientada a aspectos e oferece uma interface grafica para configuração da aplicação. O Fractal é mais limitado que o LuaSpace EPlus pois define o seu modelo de componentes proprietário, enquanto que o LuaSpace EPlus utiliza o padrão CORBA. Além disso, em termos de ferramentas de desenvolvimento, o Fractal não contempla busca de serviços, não trabalha com ontologias, nem provê mecanismos para segurança das aplicações.

O sistema *Agora* [Seacord et al 1998] descreve um mecanismo que localiza, indexa, busca e recupera componentes disponibilizados na Web. O objetivo principal do sistema *Agora* é maximizar a reusabilidade dos componentes JavaBeans e CORBA. Diferentemente do LuaSpace EPlus, o sistema *Agora* não inclui suporte à reconfiguração dinâmica e a segurança nem utiliza ontologias para evitar problemas semânticos na localização de componentes. Dessa forma, comparando-se apenas o mecanismo de busca do LuaSpace EPlus, o *Discovering Service*, com o do sistema *Agora,* observa-se que o último não implementa qualquer solução relativa ao balanceamento de carga e nem à busca assíncrona e também não esta associado a um serviço de ontologias. O propósito do LuaSpace EPlus é mais amplo do que o do Sistema *Agora*.

O *PacoSuite* [Vanderperren 2003] é um ambiente visual para composição de componentes que oferece um editor visual para documentação de componentes, o *PacoDoc*, e um ambiente de composição visual, o *PacoWire*, que permite ligar componentes para formar composição. A documentação é usada pelo PacoWire para

verificar se componentes podem ser interligados. O ambiente inclui um gerador de código que gera os códigos de composição entre os componentes. De forma semelhante ao LuaSpace, o ambiente promove a separação de conceitos através de uma extensão a POA. Diferentemente do PacoSuite, o LuaSpace Eplus não está centrado apenas na composição de componentes e em POA mas também visa beneficiar o reuso, através do serviço de seleção dinâmica baseado em ontologias.

A versão anterior de LuaSpace [Almeida et al 2004] oferecia uma interface gráfica proprietária e ferramentas para seleção dinâmica e segurança. No entanto, a seleção dinâmica não incluia a parte de ontologias. Em conseqüência, os resultados da seleção apresentavam problemas semânticos. Em relação à segurança, a versão anterior não incluía suporte para certificados X.509. Além disso, a interface gráfica para o desenvolvimento de aplicações era extremamente dependente de bibliotecas associadas com a implementação CORBA utilizada – o Mico – o que dificultava o uso do ambiente com outras implementações CORBA. O LuaSpace EPlus desenvolveu soluções para esses problemas e integrou o *Discovery* com o serviço de Ontologias além de incluir suporte para programação orientada a aspectos.

6. Conclusões

Esse trabalho apresentou o ambiente LuaSpace EPlus, um ambiente interativo no Eclipse para desenvolvimento de aplicações baseadas em CORBA que torna simples o processo de desenvolvimento, promove reuso de componentes, permite associar segurança a aplicações, separar componentes e aspectos que formam uma aplicação, além de prover um mecanismo para construção de ontologias que está integrado com o mecanismo de seleção dinâmica. O ambiente está operacional em uma das IDEs mais utilizadas no mundo – o Eclipse. O diferencial da disponibilidade do ambiente no Eclipse reside no fato que o Eclipse é uma ferramenta de desenvolvimento largamente utilizada e de fácil instalação. Portanto, eliminam-se problemas de instalação e configuração do ambiente LuaSpace e dos serviços CORBA subjacentes. Nesse contexto, o LuaSpace EPlus une a idéia de metodologias ágeis com recursos virtuais para desenvolvimento de aplicações baseadas em CORBA.

O suporte a reconfiguração dinâmica é garantido pelo fato de usar uma linguagem interpretada e dinamicamente tipada, que confere flexibilidade para reconfiguração dinâmica.

Com o LuaSpace EPlus aplicações podem ser facilmente desenvolvidas usando-se a interface gráfica através da qual se pode explorar os diferentes mecanismos oferecidos pelo ambiente e a facilidade de geração automática de código a partir de informações selecionadas na interface gráfica. Essa facilidade permite que programadores de diferentes níveis técnicos possam desenvolver aplicações facilmente. Os programadores de mais alto nível podem dispensar o uso da geração automática de código e escrever todo o código da aplicação usando o editor do ambiente ou o console Lua. Os programadores menos experientes podem tirar proveito da facilidade de geração de código a partir de informações selecionadas nas diversas janelas oferecidas pelo ambiente.

O ambiente LuaSpace e suas ferramentas estão disponíveis para download em www.consiste.dimap.ufrn.br/~andre/luaspace

Referências

Almeida, A. D., Cacho, N. and Batista, T. (2004) LuaSpace Plus: Um Ambiente Visual para Desenvolvimento de Aplicações CORBA. *Anais do VI Simposio Brasileiro de Engenharia de Software (SBES),* pp. 163-177, Brasília – DF.

Batista, T. and Rodriguez, N. (2000) "Configuração de Aplicações no LuaSpace". *Anais do 18° Simpósio Brasileiro de Redes de Computadores (SBRC 2000)*, Belo Horizonte, MG, pp 169-182.

Bernstein, P. (1996) Middleware. *Communications of the ACM*, 39(2), February 1996.

Bruneton, E., Coupaye, T. and Stefani, J.B. (2002) Recursive and Dynamic Software Composition with Sharing. Seventh International Workshop on Component-Oriented Programming (WCOP02), June 2002 - At ECOOP 2002, Malaga, Spain

Cacho, N., Batista T. and Elias, G. (2004) Um Serviço CORBA para Descoberta de Componentes. 18th Simposio Brasileiro em Eng. de Software (SBES). Brasília, DF.

Cacho, N., Batista, T. and Fernandes, F.(2005) AspectLua – A Dynamic AOP Approach. Journal of Universal Computer Science (J.UCS), Vol. 11, No. 7, pp. 1177-1197.

Cerqueira, R., Cassino, C. e Ierusalimschy R. (1999) Dynamic Component Gluing Across Different Componentware Systems. International Symposium on Distributed Objects and Applications (DOA), Edinburgh, Scotland, 1999.

Eclipse disponível em http://www.eclipse.org/

Elrad, T., Aksit, M., Kiczales, G., Lieberherr, K., and Ossher, H. (2001) Discussing Aspects of AOP. Communications of the ACM, Vol. 44, No. 10, pp 33-38, October 2001.

Ierusalimsky, R., Figueiredo, L. H., and Celes, W. (1996) Lua – an extensible extension language. Software: Practice and Experience, 26(6):635-652. 1996.

Martins, S., Cacho, N. and Batista, T. (2004) "Uma Biblioteca para Segurança de Aplicações CORBA". Aceito para publicação nos *Anais do 22° Simpósio Brasileiro de Redes de Computadores (SBRC 2004),* Gramado, RS, Maio 2004.

OMG (2004) Common Object Request Broker Architecture: Core Specification Technical Report Revision 3.0.3.

Seacord, R.; Hissan, S.; Wallnau, K, (1998) "Agora: A Search Engine for Software Components", *Technical Report CMU/SEI-98-TR-011*, August 1998.

Vanderperren, W., et al (2003) PacoSuite & JAsCo: A visual component composition environment with advanced aspect separation features. *In Proceedings of Int Conf on fundamentals of Software Engineering (FASE)*, Warshaw, Poland.