LuaSpace Plus: Um Ambiente Gráfico para Desenvolvimento de Aplicações CORBA

André Duarte de Almeida, Nélio Cacho, Thais Batista
Departamento de Informática – Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário – 59.056-300 – Natal – RN
e-mail: [andre, cacho]@consiste.dimap.ufrn.br, thais@ufrnet.br

Resumo

Este artigo descreve um ambiente visual para desenvolvimento de aplicações baseadas em CORBA chamado LuaSpace Plus. Os principais propósitos do ambiente são tornar simples o processo de desenvolvimento, promover o reuso de componentes e oferecer suporte à reconfiguração dinâmica de aplicações CORBA. Para isso, LuaSpace Plus dispõe de uma interface gráfica para desenvolvimento interativo de aplicações e ferramentas baseadas em CORBA para seleção dinâmica de componentes e para incluir funções de segurança nas aplicações. A interface inclui um browser que exibe os resultados do processo de seleção dinâmica, permite navegação em repositórios CORBA, possibilita a configuração de funções e atributos de segurança e permite a geração automática de partes do código das aplicações a partir de informações selecionadas e estabelecidas na interface gráfica.

Abstract

This paper describes a visual environment for the development of CORBA based applications named LuaSpace Plus. The main goals of this environment are to make easier the development process, to promote components reuse and to support dynamic reconfiguration of CORBA applications. In order to address these issues, LuaSpace Plus offers a graphical interface for the interactive development of CORBA based applications and tools based on CORBA to the dynamic selection of components and to include security functions in the applications. The interface includes a browser that shows the result of the dynamic selection process, it also allows to browser the CORBA repositories, it supports the configuration of security functions and attributes and it allows the automatic generation of parts of the application code from information selected and set at the graphical interface.

1. Introdução

O desenvolvimento de software baseado em componentes [8] tem sido reconhecido como uma estratégia promissora para diminuição do tempo e do custo de desenvolvimento por promover o reuso de componentes. Plataformas de middleware [1] 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. CORBA (*Common Object Broker Request Architecture*) [9] é uma plataforma de middleware que tem se destacado em relação às demais por ser uma especificação aberta, independente de fabricante e de linguagem.

Apesar das plataformas de middleware oferecerem suporte para diminuir as complexidades relativas à programação distribuída e utilização de componentes, elas não apresentam facilidades par
a a organização global de uma aplicação [4]. Para solucionar este problema o ambiente LuaSpace [2,3] adota a idéia de utilizar uma linguagem de configuração para definição da estrutura de aplicações baseadas em componentes e para dar suporte à reconfiguração dinâmica da aplicação. O ambiente explora uma linguagem interpretada e dinamicamente tipada – a linguagem Lua [5] – e um *binding* dinâmico entre Lua e CORBA – LuaOrb [6] – que possibilita o acesso dinâmico a componentes CORBA da mesma maneira que se faz acesso a objetos Lua.

O ambiente LuaSpace tem como enfoque promover o reuso de componentes CORBA conferindo flexibilidade para configuração e reconfiguração dinâmica de aplicações CORBA. Na sua versão original, a ausência de alguns mecanismos importantes para o desenvolvimento e a execução de aplicações limita a flexibilidade que o ambiente pode oferecer para programadores de aplicações e, portanto, compromete o reuso de componentes.

Uma das limitações cruciais para o reuso é o suporte restrito à seleção dinâmica de componentes. Como o reuso de componentes é o cerne do desenvolvimento baseado em componentes, para facilitar o reuso é necessário prover facilidades para que o programador possa expressar vários critérios para seleção de componentes CORBA e, inclusive, a combinação de vários critérios em uma determinada seleção. 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.

A ausência de mecanismos para garantir segurança das aplicações é outra lacuna importante, pois impossibilita o desenvolvimento de uma gama de aplicações que necessitam de segurança como aplicações de comércio eletrônico, bancárias, telecomunicações e gerenciamento de redes.

Outro aspecto limitante é o fato do LuaSpace oferecer apenas uma interface baseada em comandos (console interativo) para o desenvolvimento de uma aplicação. O programador fica restrito a utilizar editores externos ao ambiente para gerar arquivos com o código da aplicação para, posteriormente, usar o console interativo para executar o arquivo. Além disso, a ausência de um ambiente gráfico restringe a visão integrada dos mecanismos que o ambiente oferece.

Este trabalho apresenta ferramentas que foram desenvolvidas visando estender o ambiente LuaSpace em três direções: (1) oferecer um serviço CORBA para seleção dinâmica de componentes que aceita vários critérios de seleção; (2) prover mecanismos para incluir segurança nas aplicações; (3) proporcionar uma interface gráfica para o desenvolvimento interativo de aplicações baseadas em CORBA incluindo janelas para edição e execução dos programas e um *browser* que exibe os resultados da seleção dinâmica, permite navegação no repositório de interfaces e possibilita configurar atributos de segurança para as aplicações bem como gerar o código a ser inserido na aplicação.

O artigo está estruturado da seguinte forma: a seção 2 apresenta conceitos básicos de CORBA e LuaSpace. A seção 3 apresenta as extensões desenvolvidas para o ambiente visando oferecer um ambiente gráfico interativo onde as aplicações possam ser desenvolvidas, mecanismos e atributos de segurança possam ser facilmente associados às aplicações, diferentes critérios de seleção possam ser estabelecidos para se encontrar componentes e os resultados possam ser exibidos em um *browser* que ilustra as interfaces dos componentes selecionados pelo processo de seleção dinâmica. A seção 4 ilustra um estudo de caso que explora as novas ferramentas do ambiente. A seção 5 contém as conclusões.

2. Conceitos Básicos

2.1. LuaSpace

LuaSpace é um ambiente para desenvolvimento de aplicações distribuídas baseadas em componentes CORBA que integra a plataforma CORBA com a linguagem Lua. O ambiente oferece um conjunto de ferramentas baseadas em Lua com funções estratégicas para facilitar o desenvolvimento de aplicações baseadas em componentes e para promover reconfiguração dinâmica. Em LuaSpace a aplicação é escrita em Lua e pode ser composta por componentes implementados em qualquer linguagem que tenha o *binding* para CORBA. Componentes, scripts e *glue code* são os elementos que formam uma aplicação em LuaSpace.
Lua é uma linguagem interpretada, com um sistema de tipos dinâmico: variáveis não têm tipo, apenas valores estão associados a tipos. A linguagem possui algumas características não convencionais como: funções são valores de primeira classe; arrays *associativos* (chamados *tabelas*) são a única facilidade de estruturação de dados; *tag methods* é o mecanismo mais genérico para reflexão. *Tag methods* podem ser especificados para serem chamados em situações nas quais o interpretador Lua não tem como proceder.
LuaOrb é uma das principais ferramentas de LuaSpace. LuaOrb é um *binding* entre Lua e CORBA, baseado na Interface de Invocação Dinâmica de CORBA (DII), que oferece acesso dinâmico a componentes CORBA da mesma forma que se faz acesso a objetos Lua. O acesso a objetos CORBA é realizado de forma transparente. LuaOrb também usa a Interface de Esqueleto Dinâmico (DSI) de CORBA para permitir instalação dinâmica de novos objetos em um servidor em execução. Para usar um componente CORBA é necessário, primeiro, criar um *proxy* Lua usando a função *createproxy* de LuaOrb. Esta função cria um objeto Lua que representa um objeto CORBA. Através do mecanismo de *tag methods,* as operações aplicadas sobre o *proxy* são tratadas por LuaOrb que as transforma em operações sobre componentes CORBA.

2.2. Segurança e Seleção Dinâmica em CORBA

CORBA oferece um conjunto de serviços que podem ser usadas por aplicações, tais como: nomes, *trading*, eventos, segurança, transações, entre outros. Os serviços são descritos na linguagem de descrição de interfaces (IDL) de CORBA. As interfaces de todos os objetos CORBA são armazenadas no Repositório de Interfaces (RI). Para utilizar algum serviço CORBA é necessário, inicialmente, habilitá-lo. A habilitação envolve invocações às funções oferecidas pelo serviço. Portanto, para cada serviço que o programador precisa utilizar em sua aplicação, ele deve invocar uma série de comandos específicos do serviço e alheios ao domínio da aplicação.
Os serviços de *Nomes* e de *Trading* dão suporte à seleção dinâmica de componentes CORBA. O serviço de Nomes permite localizar objetos pelo nome e o serviço de *Trading* permite procurar objetos de acordo com suas propriedades. Apesar desses serviços oferecerem opções interessantes para se encontrar objetos, eles são serviços independentes. Portanto, para fazer uma consulta por nomes e propriedades, o programador tem de invocar funções de cada serviço separadamente e, posteriormente, fazer a intersecção dos resultados. A resposta obtida é um conjunto de referências de objetos. Para conhecer a interface desses objetos, o programador deverá consultar o RI. Além disso, a utilização desses serviços envolve o conhecimento de detalhes do conjunto de comandos específicos de cada serviço. Portanto, como a busca por componentes introduz complexidade, o

programador, muitas vezes, opta por desenvolver seu próprio componente ao invés de reusar componentes disponíveis.
O serviço de Segurança oferecido por CORBA – CORBASec – dispõe de um conjunto de objetos e funções de segurança que podem ser usados para proteger componentes e mensagens trocadas entre componentes em um ambiente CORBA. As funções de segurança oferecidas pelo CORBASec abrangem autenticação, controle de acesso, auditoria e não-repúdio. Além da complexidade inerente à habilitação dos serviços CORBA, o uso do serviço de segurança introduz dificuldades adicionais devido à diversidade de conceitos, objetos e funções que são definidos pelo CORBASec para prover segurança às aplicações. Por exemplo, o processo de auditoria envolve a especificação do tipo de evento (autenticação, invocação de operação, mudança de política), o estabelecimento da lista de condições (seletores) que devem ser satisfeitas para que um evento seja auditado e a determinação do sistema de avaliação da lista (validação de todas as condições da lista ou de pelo menos uma das condições). As condições podem incluir: nome da interface do objeto alvo, nome da operação, sucesso ou falha, dia da semana da chamada, etc. Portanto, cerca de 30 (trinta) linhas de código são necessárias para estabelecer uma condição muito simples de auditoria (conforme ilustrado em [7]).

3. O Ambiente LuaSpace Plus

LuaSpace Plus consiste no ambiente LuaSpace com extensões que incluem a interface gráfica, a incorporação do *Discovering Service* ao mecanismo de seleção dinâmica e uma biblioteca de segurança denominada *LOrbSec*. A extensão do ambiente em relação à seleção dinâmica e segurança explora os serviços CORBA de forma a evitar soluções proprietárias. LuaSpace Plus habilita automaticamente os serviços tornando-os operacionais para uso a qualquer momento, permitindo, inclusive, que o programador possa interagir diretamente com os serviços CORBA sem precisar habilitá-los.
A Figura 1 contém a visão geral do ambiente LuaSpace Plus incluindo a interface gráfica e a arquitetura subjacente que compreende o interpretador Lua, LuaOrb, os mecanismos de seleção dinâmica e a biblioteca LOrbSec, que oferece suporte à inclusão de segurança nas aplicações. Em LuaSpace o script de configuração da aplicação é submetido ao interpretador Lua que o executa fazendo chamadas à 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. Similarmente, quando uma função para seleção dinâmica é executada, a biblioteca para seleção dinâmica é acionada. Através de LuaOrb, LOrbSec e a biblioteca de seleção dinâmica faz chamadas a objetos, serviços e repositórios CORBA. 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 à funcionalidades não fornecidas pelo CORBASec.

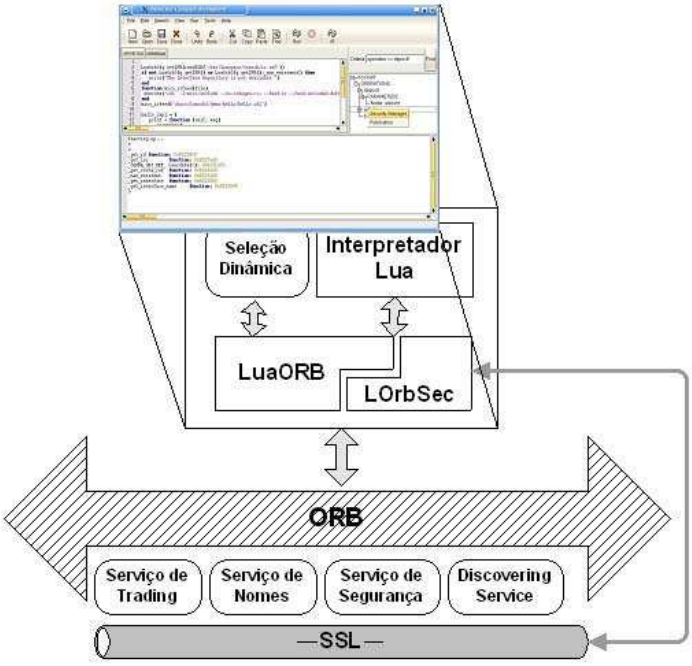

Figura 1 – Visão Geral de LuaSpace Plus

3.1. Seleção Dinâmica

O mecanismo de seleção dinâmica desenvolvido no contexto desse trabalho tem como objetivo prover uma solução integrada e fácil de usar, permitindo que programadores possam, de forma simples, localizar objetos que satisfazem determinados critérios de busca. Para isso foi implementado um novo serviço CORBA, denominado *Discovering Service* [13]. Seguindo a idéia de evitar soluções proprietárias, optou-se por aproveitar o suporte do RI e dos serviços de seleção dinâmica existentes no CORBA: Nomes e *Trading*. Desta forma o *Discovering Service* recupera as informações mantidas por esses repositórios e disponibiliza funções de inserção, remoção e modificação de serviços, além de permitir dois tipos de funções de busca: ativa e passiva. Na função ativa, chamada de *search*, o usuário informa uma condição e o serviço irá retornar os componentes que satisfazem esta condição. Na função passiva, chamada de *search_back*, o usuário informa a condição e um objeto que será usado para receber o resultado da busca. Este objeto deve implementar uma função de *callback*, que será invocada todas as vezes que novos componentes satisfizerem a condição fornecida. Este mecanismo evita que o usuário fique bloqueado aguardando a disponibilidade de um componente de interesse.

Tabela 1. Nomes de propriedades reservadas.

Nome	Descrição	Possíveis Valores
ServiceId	ID do serviço	Ex.:IDL:Account:1.0
Servicename	Nome do serviço	Ex.:Account
Versionumber	Número da versão do serviço	Ex.:1.0
Operationame	Nome da operação	Ex.:deposit
Operatiomode	Modo da operação	NORMAL ou ONEWAY
Paramname	Nome do parâmetro	Ex.:valuetotal
parammode	Modo do parâmetro	PARAM IN, PARAM OUT ou PARAM INOUT
Paramtype	Tipo do Parâmetro	tk shot, tk long, tk string, etc.
operationexceptionname	Nome da exceção emitida por uma Operação	Ex.: InvalidObjectRef
attributename	Nome de um atributo	Ex.: AccountState
attributetype	Tipo do atributo	tk shot, tk long, tk string, etc.
lastavailability	Data da ultima disponibilidade	21-03-2000-15:30:50.

A condição imposta ao *Discovering Service* respeita a gramática definida pela *CORBA Constraint Language* (CCL) e permite a composição das palavras reservadas descritas na tabela 1 com propriedades não funcionais inseridas pelos usuários. Por exemplo, a condição *(´Print´ ~ servicename) and ((operationexceptionname == ´InvalidPageType´) in (operationame == ´print´)) and (ppm > 5)* irá procurar por todos os componentes que possuam em seu nome a string *Print*, que disponibilize a operação *print* que gere uma exceção de nome *InvalidPageType* e que tenham na propriedade *ppm* um valor superior à 5.

A CCL permite ainda a aplicação de funções que modificam o resultado da busca como é o caso das funções de ordenação *max*, *min*, etc. Além destas funções, o *Discovering Service* também oferece as funções *activity* e *loadbalance*. A função *activity* seleciona apenas os componentes que estejam ativos. Por exemplo, a restrição *activity ((ppm > 10)and(servicename == 'Printer'))* retorna todas as ofertas de serviços que estejam ativas, que tenham *ppm* maior que 10 e com nome de serviço igual a *Printer*. A função *loadbalance*, por sua vez, permite realizar o balanceamento de carga entre os componentes selecionados. Para isso a função aplica um algoritmo *round-robin* para ordenar o resultado da busca, de modo que o primeiro elemento da lista seja o serviço com menor carga. Um exemplo do uso de tal função: *loadbalance((ppm > 10)and(servicename == ´Printer´))*.

```
client = Generic()
client:print("arquivo.ps")("ppm > 4")
```

Figura 2 – Programa de configuração com o conector genérico

A construção do *Discovering Service* permitiu incorporar novas características ao ambiente LuaSpace. Em termos de suporte a seleção dinâmica de objetos, a versão original [10] do LuaSpace disponibiliza o Conector Genérico e o método *search*. O conector genérico permite que se escreva uma aplicação estabelecendo apenas os serviços que se deseja usar e suas respectivas propriedades não funcionais, sem determinar o(s) componente(s) que oferece(m) tais serviços. O conector genérico seleciona dinamicamente os componentes que oferecem os serviços estabelecidos e que irão compor a aplicação. No processo de seleção destes componentes, o conector genérico usa como critério de busca a assinatura dos métodos correspondentes aos serviços a serem localizados e as propriedades não funcionais dos respectivos serviços. O conector genérico determina a execução de um dos componentes selecionados e retorna o resultado.

a. Seleção de objeto por nome: buscar um objeto cujo nome seja "print":

```
search{ {NAME = "print"} }
```

b. Seleção de objeto usando uma combinação de critérios (nome do objeto e propriedades):

```
search{{NAME = "print"}, "and",
{PROPERTIES = {constraint = "speed >= '4' and resolution = '600' "}}}
```

Figura 3 – Exemplo de especificação de critérios de seleção usando a função *search*

A sintaxe para expressar os dois critérios de busca é *c:método(parâmetros)(propriedades)*, onde c é o *proxy* do conector genérico. A figura 2 ilustra um trecho de programa de configuração que utiliza o conector genérico para selecionar um componente que tenha o método *print* e a propriedade *ppm > 4*. A utilização do conector genérico permite, portanto, que o programador realize a localização de objetos com base na assinatura de um método e também na combinação de suas propriedades não funcionais.

O outro mecanismo disponibilizado pelo LuaSpace é a função *search*. Esta função recebe como parâmetro uma lista de critérios de busca e fornece como resultado uma lista contendo as referências de objetos que satisfazem os critérios. De posse desta lista, o programador pode escolher qual(is) objeto(s) utilizar. Esta função realiza os seguintes tipos de seleção: baseado em nomes de objetos, baseado na assinatura de um método e em suas propriedades não funcionais. A figura 3 ilustra o uso da função *search*.
Para implementar tais mecanismos o LuaSpace precisava, a partir da lista de critérios, pesquisar no RI todas as interfaces que possuíssem o método desejado com sua respectiva assinatura. Em seguida, para cada interface localizada, era invocada uma consulta ao serviço de *Nomes* ou de *Trading* para localizar as ofertas de serviços associadas a cada interface IDL selecionada.
Com o acréscimo do *Discovering Service* ao LuaSpace, o conector genérico e a função *search* não precisam mais interagir diretamente com os repositórios CORBA(RI, Nomes e Trading) uma vez que essa interação é realizada pelo *Discovering.* Dessa forma as listas de critérios de busca são passadas diretamente ao *Discovering* que as processa e retorna, para o mecanismo invocado uma lista com os serviços que satisfazem os critérios fornecidos. Os critérios de busca aplicados ao conector genérico e a função *search* também sofrem um aumento na sua capacidade de seleção, já que eles passam a suportar a CCL e a usufruir das propriedades reservadas da tabela 1 e das funções *activity* e *loadbalance* fornecidas pelo *Discovering*. A função *activity*, em especial, irá minimizar em muito a utilização do mecanismo de tolerância a falhas [11] disponibilizada pelo conector genérico, uma vez que as chances de um serviço não estar disponível, apos a utilização da função *activity,* serão reduzidas. Para exemplificar a utilização do mecanismo de seleção dinâmica do LuaSpace com o suporte ao *Discovering*, reescrevemos o caso b da figura 3 da seguinte forma: *search{"(servicename =='print')and(speed >= 4)and(resolution == 600)"}*
Além de integrar o mecanismo de seleção de dinâmica do LuaSpace e aumentar sua expressividade, o *Discovering service* também permite que o usuário realize consultas ao seu repositório. Isto pode ser visto na janela superior direita da figura 4, onde ao fornecer um critério de busca o usuário obtém, do *Discovering*, uma lista de componentes contendo dados relativos a seus métodos, parâmetros de chamada, atributos e ainda podendo definir propriedades de segurança, publicação e uso.

3.2. Segurança

A extensão do ambiente em relação à segurança tem como objetivo permitir que programadores incluam segurança nas suas aplicações, explorando o suporte de CORBASec, sem a necessidade de conhecer detalhes da especificação. Para isso, desenvolvemos LOrbSec [7], uma biblioteca que permite tornar aplicações seguras abstraindo as complexidades do uso do CORBASec. Além disso, LOrbSec oferece funcionalidades adicionais não disponíveis no CORBASec: verificação de certificados e associação de nomes de domínios à principais.
LOrbSec oferece funções para autenticação, controle de acesso, auditoria e verificação de certificados. LOrbSec divide suas funções em dois níveis de funcionalidade. O primeiro é representado pela classe *LOrbSecLevel1* e o segundo por *LOrbSecLevel2*. Neste artigo descreveremos apenas as funções ilustradas na tabela 2, que fazem parte do segundo nível de funcionalidade, e que serão usadas no estudo de caso da seção 4. A descrição das demais funções pode ser obtida em [7].
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 publicas. 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 2. 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 associa-las à 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. Pode-se então chamar *LOrbSecLevel2:set_required_rights("/Dimap/Professores", "IDL:LaserPrint:1.0", "print", "SecAllRights","u")* para que o método *print* da interface *IDL:LaserPrint:1.0* possa ser invocado apenas se todos (*SecAllRights*) os direitos requisitados (u-Use) tenham sua respectiva correspondência nos direitos garantidos do objeto cliente.
A atribuição dos direitos garantidos é realizada pelo método *grant_rights*. Este método recebe como parâmetro o atributo de privilégio, uma condição para o atributo de privilégio, os direitos garantidos e um nome de *domínio*. Portanto, pode-se invocar *grant_rights("PrimaryGroupId", "Professores","u","/Dimap/Professores")* para garantir o direito de uso(*u*) para os usuários do grupo(*PrimaryGroupId) Professores* de todos os métodos do *domínio /Dimap/Professores*.

3.3. Interface Gráfica

O ambiente gráfico do LuaSpace para desenvolvimento interativo de aplicações baseadas em CORBA foi criado para tornar o processo de codificação mais rápido e eficiente. O ambiente gráfico agrega os ambientes de desenvolvimento e de execução permitindo a

edição de códigos fontes, de scripts Lua, a visualização de interfaces armazenadas nos repositórios CORBA suportados pelo *Discovering service*, a definição de funções de segurança e a execução da aplicação através do interpretador Lua. Todas estas ações podem ser realizadas de forma bastante simples via a interface gráfica. Utilizando esse ambiente gráfico algumas dessas tarefas ficam mais automatizadas e integradas permitindo que, em um mesmo ambiente, o programador possa executar suas aplicações, editá-las e ter o controle dos objetos instalados nos repositórios CORBA.

Em LuaSpace um programa de configuração é um script Lua que pode conter comandos Lua, comandos para a instalação de componentes CORBA, comandos para seleção dinâmica de componentes e comandos para inclusão de funções de segurança.

A Figura 4 ilustra a tela principal do ambiente de desenvolvimento. Na janela superior está o editor de scripts Lua que oferece as opções de edição (copiar, colar, desfazer, etc) comuns a editores. Ainda na janela superior, em seu canto direito, está o navegador do repositório do *Discovering service*, que mostra as interfaces contidas em determinado repositório. Através desse navegador é possível fazer buscas para selecionar as interfaces que satisfazem determinados critérios. A figura ilustra como critério de seleção todas as interfaces que tenham a operação 'deposit'. As interfaces ilustradas na figura atendem a tal critério.

O *Discovery service* executa em paralelo à interface gráfica e trata todas as requisições de busca enviadas pela interface retornando o vetor de objetos que descreve as interfaces que satisfazem o critério de seleção, todos os seus métodos, atributos e demais informações. A partir desse vetor, a árvore de navegação das interfaces é montada e exibida na janela, como ilustra na figura 4.

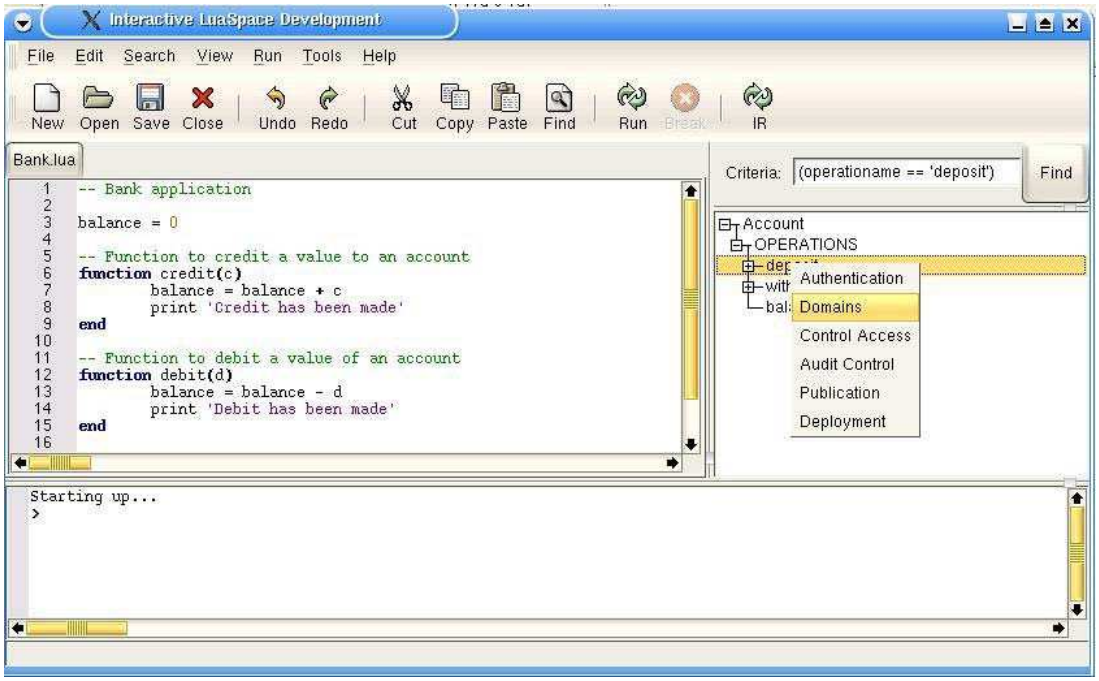

Figura 4 – Interface Gráfica do Ambiente LuaSpace Plus

A janela inferior é a do console do interpretador Lua. O usuário pode editar seu script na janela superior e determinar sua execução com uma simples clicar de mouse no botão *Run*. O sinal de ">" no console do interpretador Lua funciona como um *prompt* para o usuário indicando que está pronto para receber um novo comando. Quando algum código está em

execução, o botão *Break* é habilitado para permitir que o usuário interrompa o processamento.
A interface gráfica é continuamente atualizada durante a execução de um script para evitar que o usuário tenha a impressão de que a execução do programa está travada. O ambiente verifica, continuamente, se há algum evento da interface gráfica pendente e, se houver, o evento pendente é tratado e a execução do script prossegue. Dessa maneira, é possível realizar outras tarefas enquanto um script está em execução, por exemplo, editar arquivos, localizar interfaces, etc.
O ambiente de desenvolvimento gráfico foi implementado em C++, utilizando para a parte gráfica a biblioteca Gtk+ por oferecer um completo conjunto de componentes, além do componente de edição de código GtkScintilla que provê uma sofisticada gama de recursos como *undos* e *redos* ilimitados, estilo de sintaxe, coloração de código, dentre outros. Atualmente esse está operacional no ambiente Debian 8.0 com Lua 4.01 e LuaOrb 2.0b utilizando a implementação do CORBA MICO 2.3.6.

4. Estudo de Caso

Para ilustrar parte das funcionalidades fornecidas pelo ambiente, discutiremos, nesta seção, um estudo de caso que explora as facilidades de gerenciamento de requisitos de segurança e produção de código aplicadas a um sistema de impressão. O sistema em questão é composto por vários servidores de impressão que disponibilizam a operação *print* para os clientes que devem ser inicialmente autenticados e validados para utilizar este método. A Figura 5 descreve a interface do servidor de impressão que será implementado. 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 LaserPrint{
2       void print(in string filename);
3        void configure(in string papertype);
4 };
```

Figura 5 – Interface do servidor de impressão

O ambiente gráfico do LuaSpace Plus permite o acompanhamento do processo de desenvolvimento desde a criação da interface, passando pela publicação, pela determinação das prerrogativas de segurança, chegando até a utilização por parte do cliente. Um exemplo disso pode ser visto na figura 6 onde, após a edição da interface, o usuário pode disponibilizar sua implementação através do menu *Tools* na opção *Publication*. Neste momento será aberta a tela ilustrada na figura 6. Nesta tela, para cada campo que é fornecido, é gerado um código Lua correspondente. Portanto, quando o usuário determina que irá implementar os métodos *print* e *configure* através da indicação do valor *Publicate* para *yes*, a interface reconhece, por sua vez que deve produzir o código relativo as linhas 18 a 26 da figura 8. Da mesma forma, quando o usuário escolhe para o campo *Repository*, o valor *Discovery* e insere as propriedades *ppm* e *color* na tabela *Properties*, ele esta indicando à interface que produza o código relativo as linhas 28 e 29 da figura 8.

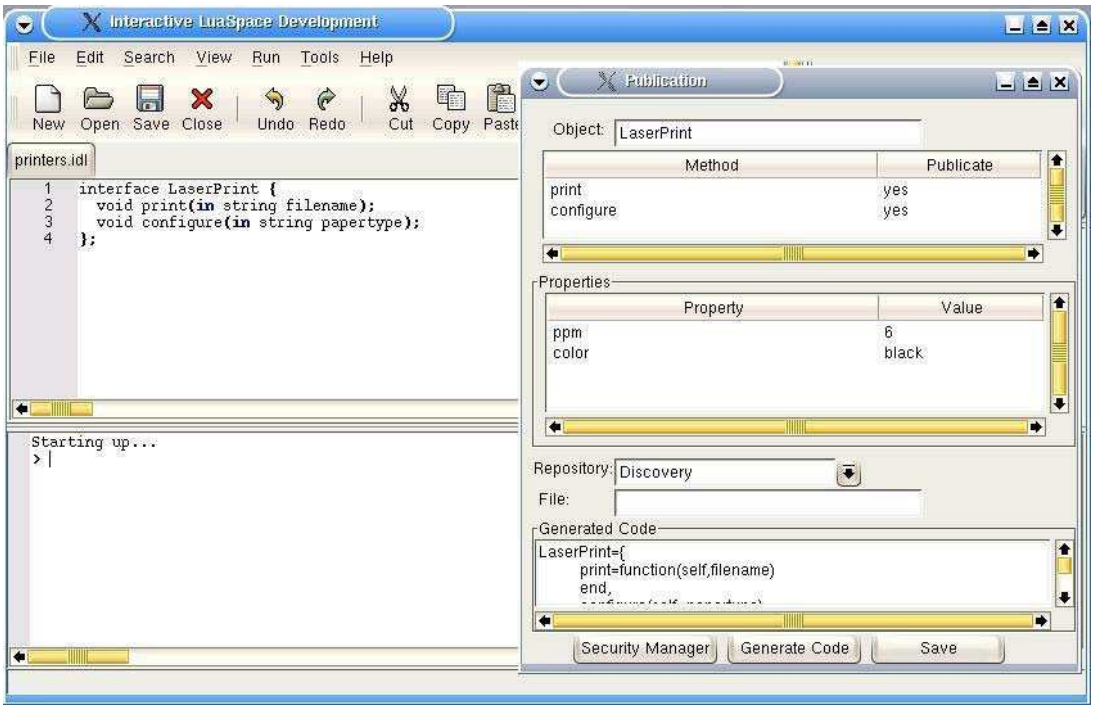

Figura 6 – Publicação da interface do servidor de impressão

A próxima fase, após a implementação do serviço, consiste na definição das prerrogativas de segurança. Esta fase é ilustrada pela figura 7. A tela de configuração das prerrogativas de segurança é composta por quatro abas, sendo elas: *Authentication, Domains, Access Control e Audit Control*. Em cada uma dessas abas o usuário determina um requisito de segurança, por exemplo, na aba *Authentication* o usuário pode determinar a criação de novos certificados, como o que descreve, nas linhas 1 a 9 da figura 8, a criação de um certificado para o usuário *Thais*. Através dessa aba é possível determinar também quais os arquivos que serão usados para realizar a autenticação do objeto servidor. O código gerado para isso é descrito na linha 11. Na aba *Domain*, o usuário pode construir de forma simples e direta a hierarquia de domínios de sua aplicação e ao final, a interface pode gerar o código lua correspondente a hierarquia produzida. Esta ação é descrita pelas linhas 13 e 14 da figura 8. Na aba *Access Control*, o usuário deve definir os direitos requeridos e garantidos para execução de cada um dos métodos da interface. Para determinar os direitos requeridos, o usuário utiliza-se da sub-aba *Required Rights* e informa quais os métodos que sofreram controle do mecanismo de controle de acesso do LOrbSec. A determinação desses requisitos é ilustrada pelo código da linha 16 na figura 8. Uma vez determinado os direitos requeridos, é necessário indicar os usuários ou grupo de usuários que irão ter acesso aos métodos protegidos. Para isso, a sub-aba *Granted Rights* permite que o usuário visualize os métodos protegidos, como é o caso do método *print* na figura 7, e a hierarquia de domínios no qual o mesmo está associado, permitindo assim uma maior facilidade e segurança na determinação dos direitos garantidos. O código relativo a determinação dos direitos garantidos é ilustrada na linha 17 da figura 8.

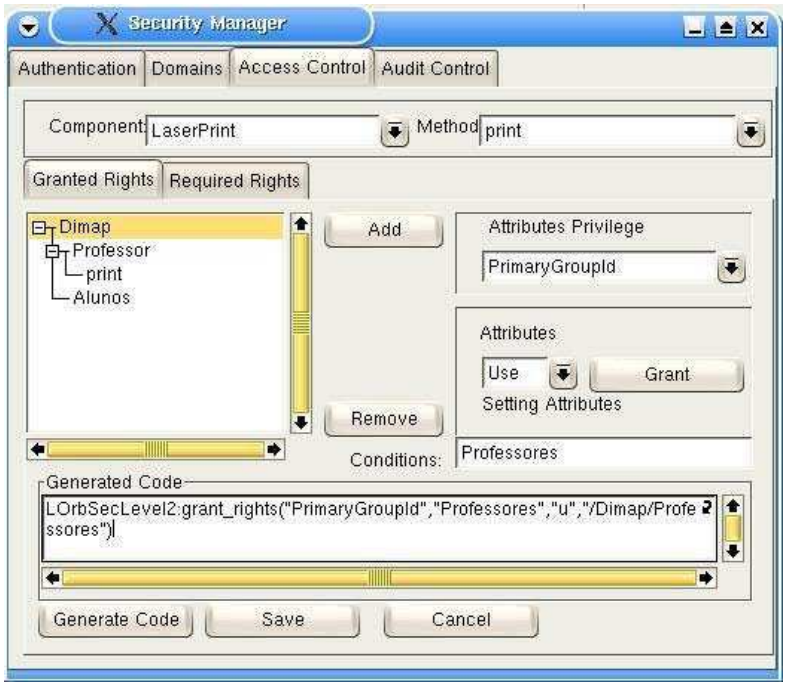

Figura 7 – Definição das prerrogativas de segurança

Após a configuração das prerrogativas de segurança o usuário deve clicar no botão *Generate Code* para gerar o código relativo às configurações realizadas. O código integral pode ser visto na figura 8.

```
ProfCert=SecurityLevel2:create_certificate("BR","RN","Natal","UFRN","Professores","Thais","
thais@dimap.ufrn.br")

3  writeto("profCert.pem")
4  write(ProfCert.get_cert())
5  writeto()

7  writeto("profKey.pem")
8  write(ProfCert.get_key())
9  writeto()

11 SecurityLevel2:authenticate("ServCert.pem","ServKey.pem")

13 LOrbSecLevel2:create_domain("/Dimap/Professores")
14 LOrbSecLevel2:create_domain("/Dimap/Alunos")
 LOrbSecLevel2:set_required_rights("/Dimap/Professores","IDL:LaserPrint:1.0","print",
              "SecAllRights","u");
17 LOrbSecLevel2:grant_rights("PrimaryGroupId","Professores","u",
               "/Dimap/Professores")
18 LaserPrint={
       print=function(self,filename)
              printFile()
21     end,

23     configure=function(self,papertype)
              conftype()
       end
26  }

28  properties1 = {ppm = 6, color = "black"}
29  source_server1 = lo_createservant(LaserPrint,"IDL:LaserPrint:1.0", properties1)
```

Figura 8 – Código gerado após as configurações realizadas

Com a execução por parte do usuário do código da figura 8, o serviço LaserPrint já encontra-se disponível para ser localizado e reutilizado. Os passos necessários para isso são mostrados na figura 9. Nesta figura o usuário insere o critério *(operationame* ==

'print')and (ppm > 4) e em seguida clica no botão *Find*. Neste momento, o usuário clica com o botão direito em cima do método *print* da interface *LaserPrint* e escolhe no pop-menu a opção *Use* exibindo a janela *Use.* Nesta janela, o usuário deve indicar se irá utilizar o conector genérico, através da opção *Use Generic Conector*, e determinar qual o critério aplicado ao conector. Por fim o usuário deve determinar o valor dos parâmetros do método selecionado. O código relativo a definição desses valores é representado pelas linhas 3 e 4 da figura 10.

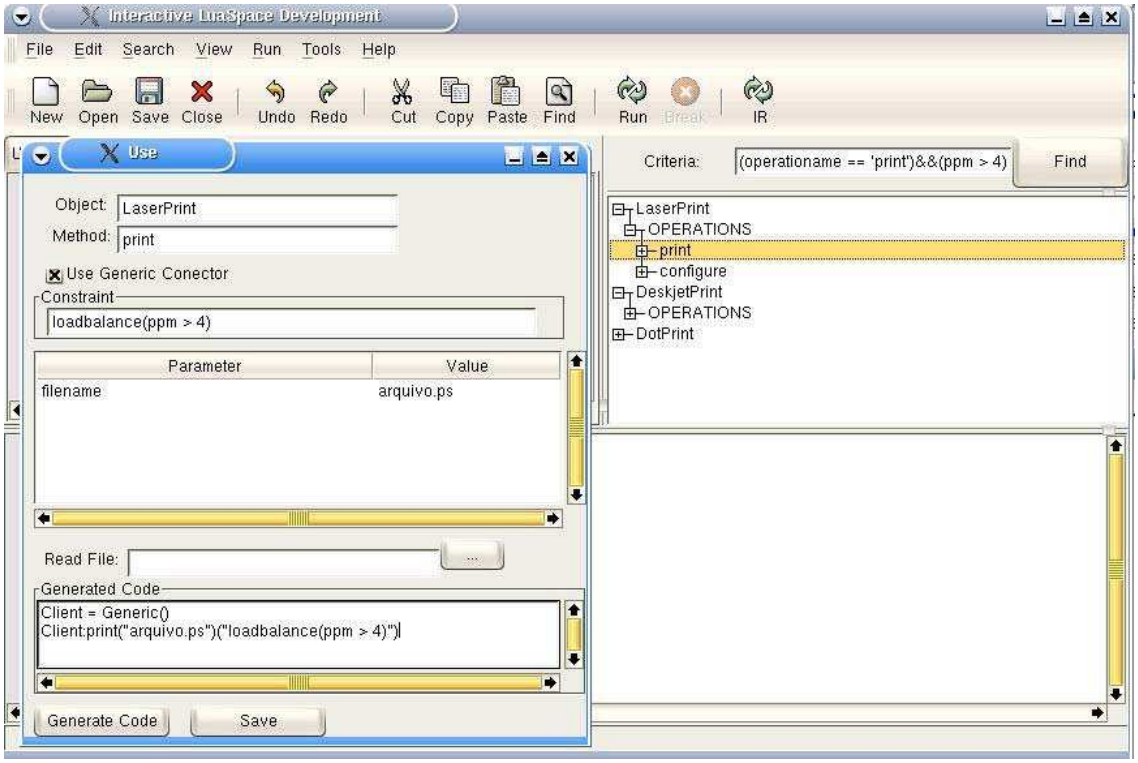

Figura 9 – Utilizando implementação do servidor de impressão

```
1 AuthStatus = SecurityLevel2:authenticate("profCert.pem" ," profKey.pem ")
2 if (AuthStatus:authentication_state()=="Success") then
        Client = Generic()
        Client:print("arquivo.ps")("loadbalance(ppm > 4)")
5 else
        print("Authentication_Error")
7 end
```

Figura 10 – Código que utiliza a implementação do serviço de impressão

5. Conclusões

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. Segurança é um dos aspectos primordiais uma vez que os componentes que formam uma aplicação estão em um ambiente distribuído, o que aumenta a possibilidade de violações de segurança.

Neste artigo apresentamos o ambiente LuaSpace Plus, um ambiente visual de desenvolvimento de aplicações baseadas em CORBA que torna simples o processo de

desenvolvimento, promove o reuso de componentes e permite associar segurança às aplicações, além de oferecer uma interface gráfica interativa que torna o processo de desenvolvimento mais rápido e automatizado. Programadores podem desenvolver aplicações facilmente utilizando a interface gráfica através da qual se pode explorar os mecanismos de seleção dinâmica e de segurança 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.

De forma a evitar soluções proprietárias, os mecanismos desenvolvidos nesse trabalho aproveitam o suporte dos serviços CORBA e dos seus repositórios de meta-informação. A interação dos mecanismos de LuaSpace com os serviços CORBA é transparente para o programador. O *Discovering Service* oferece uma maneira simples e uniforme de estabelecer diversos critérios de seleção para uma busca. As interfaces dos componentes selecionados pelo serviço são exibidas no *browser* através do qual o programador pode visualizar todos os detalhes da interface do componente. LOrbSec oferece um conjunto de funções para tornar aplicações seguras eximindo o programador da necessidade de conhecer a complexa estrutura do CORBASec. Para minimizar o esforço de programação, funções e atributos de segurança podem ser configurados via uma interface gráfica que também gera o código correspondente.

Em termos de trabalhos relacionados, podemos citar o sistema *Agora* [12], que 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 Plus, o sistema *Agora* não inclui suporte à reconfiguração dinâmica e segurança. Dessa forma, comparando-se apenas o mecanismo de busca do LuaSpace Plus, o *Discovering Service*, com o do sistema *Agora,* observa-se que o último não implementa qualquer solução relativa balanceamento de carga e nem à busca assíncrona.

Referências

1. Bernstein, P. (1996) Middleware. Communications of the ACM, 39(2), February 1996.
2. 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, Maio 2000, pp 169-182.
3. Batista, T.; Cerqueira, R. and Rodriguez, N. (2003) "Enabling Reflection and Reconfiguration in CORBA". Proceedings of the 2nd Workshop on Reflective and Adaptative Middleware - ACM/IFIP/USENIX International Middleware Conference, ISBN 85-87926-03-9, Rio de Janeiro, RJ, Junho 2003., pp 125-129.
4. Issarny, V. and Bellissard, L. and Riveill, M. and Zarras, A. (1999) "Component-Based Programming of Distributed Applications". Advances in Distributed Systems pp.327-353
5. Ierusalimschy, R., Figueiredo, L. H. and Celes, W. (1996) "Lua – an extensible extension language". Software: Practice and Experience, 26(6):635-652.
6. Cerqueira, R., Cassino, C. and Ierusalimschy, R. (1999) "Dynamic Component Gluing Across Different Componentware Systems". In International Symposium on Distributed Objects and Applications (DOA´99), 362-371, Edinburgh, Scotland, September 1999. OMG, IEEE Press.

7. Martins, S., Cacho, N. and Batista, T. (2004) "Uma Biblioteca para Segurança de Aplicações CORBA". Anais do 22o. Simpósio Brasileiro de Redes de Computadores (SBRC'2004), Gramado, RS, Maio 2004, ISBN 85-88442-79-5, pp. 511-524.
8. Szyperski, C. (1998) Component Software: Beyond Object-Oriented Programming. Addison-Wesley.
9. Tari, Z. and Bukhres, O. (2001) Fundamentals of Distributed Object Systems – The CORBA perspective. John Wiley & Sons.
10. Batista, T.; Morais, J.; Carvalho, M. and Teixeira, W. (2000) "Seleção Dinâmica de Objetos Distribuído no Ambiente LuaSpace". Anais do 20° Simpósio Brasileiro de Redes de Computadores (SBRC 2002), Búzios, RJ, Maio 2002, pp 703-718.
11. T. Batista and M. Carvalho. "Component-Based Applications: A Dynamic Reconfiguration Approach with Fault Tolerance Support". In Software Composition Workshop (SC) - affiliated to European Joint Conferences on Theory and Practice of Software (ETAPS), Grenoble - FR, April 2002. Published in Electronic Notes in Theorical Computer Science, Vol. 65, Number 4, 2002. http://www.elsevier.nl/locate/entcs/volume65.html
12. Seacord, R.; Hissan, S.; Wallnau, K, "Agora: A Search Engine for Software Components", Technical Report CMU/SEI-98-TR-011, August 1998.
13. Cacho, N. and Batista, T. (2004) "Um serviço CORBA para Descoberta de Componentes". Aceito para publicação nos Anais do 18° Simpósio Brasileiro de Engenharia de Software (SBES'2004), Brasília, DF, Outubro 2004.