Os processos de compilação e execução em AURORA

Luiz Carlos Zancanella[1]
Dr. Philippe O. A. Navaux[2]

Pós-Graduação em Ciência da Computação
Instituto de Informática - UFRGS/RS
Porto Alegre/RS - BRASIL

Resumo

Este artigo apresenta um novo enfoque para compilação e execução de sistemas orientados a objetos. O enfoque proposto permite que objetos, ou grupos de objetos, sejam isoladamente instanciados a partir de classes previamente conhecidas, ou dinamicamente configuradas, pelo sistema. O modelo proposto esta inserido no projeto Aurora, que visa a construção de um sistema operacional orientado a objetos para execução em máquinas multiprocessadoras.

Abstract

This paper describes a new approach to compiling and running object-oriented system. The mechanism proposed allows objects or groups of objects to be instantiated in separate from classes known by system or dynamicaly defined. The model is part of the Aurora project which goal is to build a multiprocessor object-oriented operating system.

1 Introdução

A evolução da engenharia de software, particularmente em direção ao modelo orientado a objetos, tem contribuído nos últimos anos para o surgimento de uma nova geração de sistemas operacionais [2], [5], [6], [10], mais dinâmicos, mais flexíveis e capazes de suportar de forma transparente a presença do processamento cooperativo, distribuído ou não, heretogêneo ou não. Evidentemente, existem outras influências envolvidas neste processo, como por exemplo, a evolução do hardware e da tecnologia de comunicações.

A forte ascendência da engenharia de software sobre esta nova geração de sistemas operacionais, pode ser justificada pelo fato de que o paradigma de objetos, introduz conceitos

[1]Doutorando do CPGCC/UFRGS; Professor Afastado da UFSC/SC, E-Mail: zancanel@inf.ufrgs.br

[2]Professor CPGCC/UFRGS, E-Mail: navaux@inf.ufrgs.br

e abstrações, que encapsulam naturalmente dentro do próprio modelo, muitos dos problemas envolvidos no projeto de sistemas operacionais, tais como; identificação, sincronização, atomicidade e proteção.

Outro fator que esta influenciando no surgimento de uma nova geração de sistemas operacionais é o aspecto qualitativo[8], principalmente no que diz respeito a ambientes de programação e interfaces. Parece claro que, apesar de desenvolvidos de forma independentes, sistemas operacionais e linguagens de programação estão fortemente relacionados e que, a crescente introdução de recursos às linguagens, antes somente providos pelos sistemas operacionais e vice-versa, demonstra a necessidade de se prover modelos capazes de suportar abstrações conceituais independente de nível, recursos ou ambientes.

Engenheiros, projetistas e pesquisadores, tem frequentemente pesquisado por modelos, linguagens e características que facilitem os processos de desenvolvimento e programação, de modo a incrementar a produtividade do programador. O modelo orientado a objetos tem sido apresentado correntemente como um enfoque promissor e capaz de municiar programadores com poderosas técnicas para escrever, estender e reusar programas rápida e facilmente.

Projetos recentes deram origem a várias linguagens, entretanto algumas características, quanto implementadas em linguagens puramente orientadas a objetos não tem apresentado resultados satisfatórios, executando de forma muito mais lenta que suas implementações em linguagens tradicionais. Por exemplo, a implementação mais eficiente para troca de mensagens em Smalltalk-80 executa numa razão 1/10 menor que uma implementação C.

Esta perda de eficiência, segundo Chambers[3] é ocasionada em grande parte pela carência de tecnologia de implementação adequada ao modelo de objetos, fato que tem forçado muitas das linguagens orientadas a objetos serem projetadas de forma híbrida, ou mesmo traduzidas em alto nível para as tradicionais linguagens não orientadas a objetos na busca de implementações eficientes.

Entretanto, o problema não está localizado somente no nível de implementação das linguagens, mas principalmente relacionado com a atual tecnologia de gerenciamento de objetos, visto que, atualmente os sistemas operacionais estão voltados para o suporte ao modelo de processos e conseqüentemente não possuem a habilidade para manter e gerenciar objetos eficientemente. Deve-se observar que sob o ponto de vista do sistema operacional, objetos são entidades de granulosidade mais leves que processos. Por exemplo um objeto quando ativado não exige necessariamente a presença de um contexto e/ou de uma área de trabalho.

Deste modo, a pesquisa de Aurora está direcionada para a busca de um modelo apropriado ao gerenciamento de objetos e capaz de proporcionar a existência de um modelo uniforme, tanto para o nível do sistema operacional como para o nível da aplicação.

Neste contexto, o modelo adotado por Aurora, e o fato de ser um sistema orientado a objetos em todos os níveis, cria condições para uma maior integração entre o sistema operacional e as linguagens. A conseqüência, do modelo estrutural implementado por Aurora, é o surgimento de um novo conceito de compilação e execução para modelos orientados a objetos, capazes

de permitir que os sistemas sejam compostos de forma dinâmica e interativa ao longo de sua execução.

2 Aurora e o Paradigma de Objetos

Devemos observar que o paradigma de objetos, semelhantemente a outros modelos, engloba dois aspectos distintos: o *modelo computacional* e o *modelo de projeto* como filosofia de desenvolvimento.

O *modelo de projeto*, é centrado na identificação e organização de conceitos no domínio da aplicação. Superficialmente podemos dizer que modelar um problema, segundo o enfoque de objetos, implica na decomposição deste problema em termos de objetos e na construção de uma hierarquia de classes de objetos, que por sua vez compõem propriedades comuns a subclasses de objetos.

Por outro lado, *o modelo computacional* é representado por um conjunto de objetos cooperando, através de mensagens, na realização de uma determinada tarefa. Neste modelo, preconizado pelo paradigma, estão ocultas características importantes, tais como: instanciação, uniformidade, proteção e paralelismo.

Instanciação reflete o nascimento do objeto, a partir do qual o objeto passa a exibir o comportamento que lhe é estabelecido pela classe a qual o mesmo está associada. A uniformidade, implica em que objetos e mensagens sejam os únicos componentes do modelo, isto é, todos os tens no sistema são objetos, desde simples valores numéricos (tais como inteiros) até entidades mais complexas abstraídas do mundo real, por outro lado, todas as operações desejadas são deflagradas através de mensagens de um objeto para outro. A característica de proteção, garante que todo o processamento sobre um objeto ocorra dentro do objeto, sendo somente ativado através de mensagens ao objeto. O paralelismo reflete o próprio paradigma, que se propõe a modelar objetos que no mundo real são naturalmente paralelos (e distribuídos).

Deve-se observar, que apesar do modelo de objetos ter nascido no escopo das linguagens de programação, orientação a objetos é fundamentalmente uma forma de desenvolvimento e não uma técnica de programação. O desenvolvimento orientado a objetos é um processo conceptualmente independente da linguagem de programação até o estágio de implementação, que pode ser feito através de uma linguagem orientada a objetos ou não.

Evidentemente que a utilização de uma linguagem orientada a objetos na representação final da aplicação, elimina muitas das restrições impostas pelas linguagens tradicionais, visto que, linguagens orientadas a objetos implementam suporte a tipos abstratos de dados, herança e troca de mensagens, diretamente dentro da linguagem.

Dentro deste contexto e respaldado pelo parecer de Goldberg[4] de que, *no modelo de objetos executar um sistema é algo tão simples como, criar objetos e disparar uma mensagem*, Aurora propõe que o ambiente de desenvolvimento e a interface sejam baseadas exclusivamente no modelo de objetos, de modo que as únicas entidades visíveis ao usuário sejam os objetos (e classes de objetos).

A implicação direta da adoção desta filosofia sobre o conceito de compilação, é que a função básica do compilador, torna-se compilar classes separadamente (meta objetos em Aurora). Estes meta objetos serão conhecidos pelo sistema operacional através de bibliotecas. Uma vez conhecidos pelo sistema, os meta objetos podem ser referenciados em tempo de execução, para instanciação de objetos, ou para ativação de seus métodos.

Por outro lado, a implicação direta do modelo sobre o conceito de execução, é que o sistema operacional não recebe mais programas prontos e montados através de processos de *link* edição, mas sim, instanciações e ativações de objetos, oriundas de outros objetos ou diretamente do usuário que interagem através da interface.

Deste modo Aurora procura refletir no *modelo computacional* a premissa do paradigma de orientação a objetos, de que o processamento de informações deva ser representado por um conjunto de mensagens fluindo entre objetos executando de forma paralela. Este processo é viabilizado através do modelo implementado por Aurora, que permite que o modelo computacional seja definido através da interface de forma interativa e dinâmica.

3 O Sistema Operacional Aurora

Visando refletir o modelo preconizado pelo paradigma de orientação a objetos, Aurora foi projetado para explorar o paralelismo, tanto a nível do sistema como a nível da aplicação. O modelo de programação concorrente orientada a objetos [1],[7],[9],[11]é utilizado como base, de modo que uma coleção de objetos é distribuída entre os processadores que interagem através de mensagens, independentemente da localização e do estado dos mesmos.

Para implementar este modelo, Aurora provê um conjunto de facilidades e abstrações procurando viabilizar um tratamento uniforme aos objetos, independente de estarem os mesmos implementando aplicações dos usuários ou serviços do sistema. Tais abstrações são utilizadas para incorporar em tempo de execução: criação dinâmica de objetos, acoplamento dinâmico, gerenciamento de mensagens, pesquisas a métodos e coleta de lixo.

O modelo estrutural, sobre o qual o sistema operacional Aurora é implementado, está baseado no modelo introduzido por Apertos[10]. O princípio de Apertos é que o nível de objetos e o nível de abstração das linguagens podem ser separadamente descritos e representados dentro da mesma estrutura. Tal princípio permite que Aurora utilize o conceito de separação entre

objetos e meta objetos, onde o objeto representa o estado do objeto (depositório de dados), enquanto o meta objeto define a abstração da classe, ou seja define semântica e comportamento.

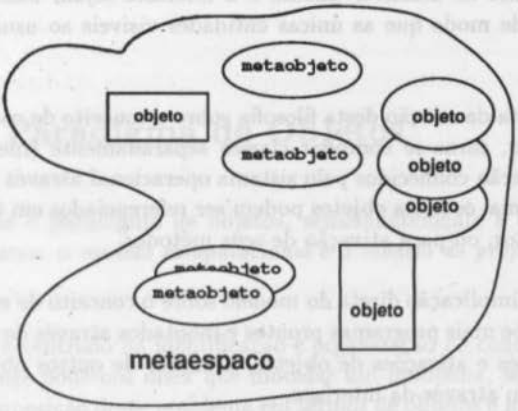

Figura 1: Visão Conceptual do modelo Aurora

A figura 1 apresenta uma visualização do modelo estrutural empregado, onde a instanciação de um objeto é suportada em um meta espaço, que pode ser visto como um sistema operacional otimizado para execução do objeto. O meta espaço é composto por um, ou mais, meta objetos que atribuem aos objetos um conjunto de abstrações, que definem a semântica do comportamento dos objetos pertencentes a este meta espaço. Os serviços do sistema operacional são implementados através de metaobjetos, cada um dos quais fazendo parte de algum meta espaço dentro do sistema.

A utilização do conceito de separação entre objetos e metaobjetos, acrescida da habilidades dos metaobjetos representarem a abstração da classe, acrescenta a Aurora a habilidade para implementar o conceito de herança dinâmica de classe[13]. Isto permite que a construção hierárquica das classes seja realizada dinamicamente e em tempo de execução e não mais de forma estática e em tempo de compilação como ocorre em Apertos.

Note que tradicionalmente o mecanismo de herança, ainda que herança múltipla seja suportada, somente está disponível em tempo de compilação, isto é, a hierarquia da classe é destruída quando o objeto é compilado. A habilidade de Aurora de suportar herança dinâmica de classe introduz as classes de capacidade evolutiva, ou seja, permite que classes de objetos adquiram dinamicamente uma nova semântica e novas propriedades de execução.

Aurora implementa um modelo uniforme tanto para o sistema como para as aplicações do usuário, de modo que o suporte ao modelo estrutural, nível mais básico do sistema, também seja implementado através de metaobjetos. Este contexto é formado por um conjunto de meta espaços que implementa, basicamente, mecanismos (metaobjetos) para gerenciamento de meta espaços, mecanismos para permitir migração de objetos, mecanismos para suporte à compilação

e mecanismos para suporte à herança dinâmica.

A figura 2 apresenta uma visão simplificada do modelo implementado, onde o suporte ao gerenciamento de meta espaços constitue o topo da hierarquia, a partir do qual sucessivos meta espaços implementam mecanismos, serviços do sistema operacional e aplicações do usuário, que compartilham facilidades comuns tais como migração de objetos.

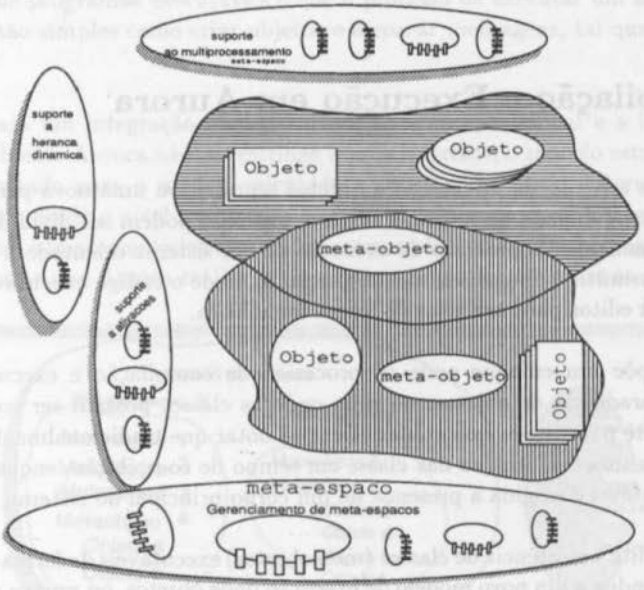

Figura 2: Visão Simplificada da Implementação de Aurora

Deve-se notar, que o mecanismo básico para construção do sistema operacional Aurora é a *migração de objetos*. Migração de objetos é definido de tal maneira que um objeto troca de meta espaço quando ele necessitar de algum serviço suportado em outro meta espaço. Por exemplo, um objeto pode migrar para um meta espaço que representa memória secundária quando é para ser armazenado em disco.

Outras características estão presentes em Aurora[12]. Por exemplo, o suporte a livre granulosidade, tal facilidade é oriunda do fato de que o modelo proposto é um ambiente completamente uniforme e consistente para todas as entidades. Outra característica é a transparência ao escalonamento dos processadores, visto ser um dos objetivos a busca de alto desempenho na utilização das arquiteturas multiprocessadoras, Aurora deve adotar um cuidadoso critério na distribuição dos objetos, de modo a minimizar referências remotas, por serem tais referências em geral mais dispendiosas que as referências locais e, ainda que imagina-se o usuário com conhecimento da arquitetura, a distribuição dos objetos é totalmente transparente ao usuário, sendo os meta espaços gerados com base no conhecimento do tipo de invocação e da carga do sistema.

Uma característica marcante de Aurora é ser um sistema puro, ou seja, apresentar exclusivamente suporte para linguagens orientadas a objetos. A adoção deste modelo além de permitir um tratamento uniforme, tanto para o sistema como para as aplicações do usuário, facilita a integração entre o sistema operacional e as linguagens de programação, apesar de originalmente desenvolvidos de forma independente.

4 Compilação e Execução em Aurora

Ainda que o advento da orientação a objetos representou uma nova perspectiva para modelagem e desenvolvimento de software, onde os sistemas podem ser divididos em vários componentes independentes, o processo de execução de um sistema orientado a objetos continuou semelhante às primitivas linguagens de programação, onde o código executável é montado pelo compilador/*link* editor para executar de forma monolítica.

Aurora propõe um enfoque onde os processos de compilação e execução estejam mais próximos do paradigma de objetos, ou seja, onde as classes possam ser compiladas e existirem isoladamente para serem executadas. Deve-se notar que tradicionalmente os compiladores destroem a estrutura hierárquica das classe em tempo de compilação, enquanto, para geração de código executável é exigida a presença de um corpo principal do sistema.

Aurora permite a existência de classes (metaobjetos) executáveis de forma isolada. A adoção deste modelo conduz a um novo modelo de execução onde objetos, ou grupos de objetos, podem ser isoladamente instanciados a partir da interface do sistema, ou através de outros objetos.

Note que, segundo o *modelo de projeto*, o resultado do processo de modelagem de um sistema orientado a objetos é um conjunto de classes, que representam os conceitos (entidades reais e abstratas) existentes no domínio do problema. No modelo Aurora a representação de tais classes (códigos fontes) será realizada através de alguma linguagem orientada a objetos e armazenada em objetos que representam bibliotecas de classes.

Em Aurora a efetivação do processo de compilação sobre uma classe produz como resultado uma entidade fundamental do *modelo computacional* de Aurora chamado meta objeto. Isto é, a partir do momento em que uma classe é compilada, o meta objeto resultante passa a ser conhecido pelo sistema através da inserção do mesmo no universo de metaobjetos disponíveis. Deve-se notar que os metaobjetos definem a semântica e o comportamento dos objetos, isto é, objetos são instanciados associados a metaobjetos e passam a exibir o comportamento por eles estabelecido.

Entretanto, a disponibilidade deste universo de metaobjetos não é suficiente para representar o comportamento de um sistema, sendo necessário estabelecer-se um modelo computacional, isto é, definir uma lógica para o modelo executável. Em Aurora este processo é realizado através

da definição de um conjunto de instruções, tipicamente instanciações e/ou ativação de objetos, que visam produzir um sistema completo.

Aurora permite que este processo seja representado diretamente pela interface do sistema através de uma "linguagem de comandos" projetada especialmente para manipulação de objetos. É importante o leitor perceber que a criação de objetos, no modelo tradicional, somente ocorre através de programas executáveis e que o processo de executar um sistema em Aurora tornou-se algo tão simples como criar objetos e disparar mensagens, tal qual preconizado por Goldberg.

Para que haja um integração efetiva entre o sistema operacional e a linguagem de programação o ambiente Aurora não se restringe a uma interface; o modelo estrutural empregado é totalmente voltado para o modelo orientado a objetos, de modo a permitir que todos os níveis do sistema, desde o nível mais baixo do sistema operacional até a interface com usuário, sejam projetados e implementados através da mesma abstração. Tal homogeneidade, além da integração, permite que Aurora realize um gerenciamento eficiente sobre os objetos.

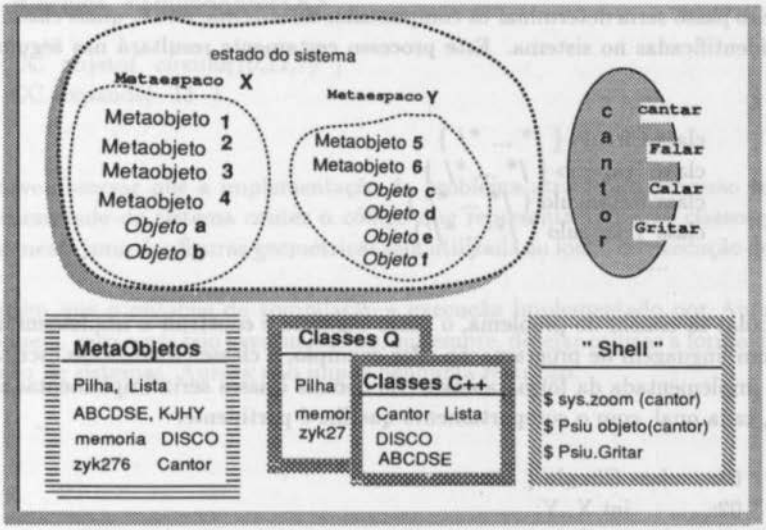

Figura 3: Representação da Interface Aurora

Um exemplo claro desta integração é a possibilidade do usuário criar objetos através da interface e utilizar um meta objeto, definido e implementado em uma linguagem de programação, para estabelecer o comportamento do objeto. Note que a utilização dos recursos implementados pelos metaobjetos pode ocorrer diretamente pelo sistema ou através de objetos criados em uma linguagem de programação, independente dos mesmos estarem armazenados em dispositivos secundários (em bibliotecas), ou ativos compondo meta espaços no processador local ou remoto.

Evidentemente que Aurora deve prover um modelo de "linguagem de comandos" com recursos mais poderosos que a simples instanciação e ativação de objetos. Por exemplo, são providos recursos para instanciação e ativação de objetos em paralelo, visto que Aurora é projetado para arquiteturas multiprocessadoras. Outro exemplo do poder da interface, é a existência de herança dinâmica de classes, permitindo que os componentes (classes) do sistema sejam modelados ao longo de sua execução.

A figura 3 apresenta uma possível visualização do ambiente Aurora, mostrando bibliotecas de classes de objetos escritas em linguagens orientadas a objetos, a biblioteca de metaobjetos disponíveis, uma visão simplificada do estado do sistema (conjunto de metaespaços) e a interface através da qual o usuário interage com o sistema, instanciando e/ou enviando mensagens aos objetos.

Para exemplificar a forma como um sistema é implementado em Aurora, vamos supor a necessidade de se construir um sistema para representar uma figura geométrica entre o seguinte conjunto de figuras: {círculo, triângulo, retângulo, trapézio, ...}.

O primeiro passo seria determinar os componentes do sistema, isto é, quais classe de objetos podem ser identificadas no sistema. Este processo certamente resultará nas seguintes classes do tipo:

```
class Círculo { /* ... */ }
class Trapézio { /* ... */ }
class Retângulo { /* ... */ }
class Triângulo { /* ... */ }
.....
```

Identificadas as classes do problema, o passo seguinte é construir e implementar as classes através de um linguagem de programação. Por exemplo, a classe círculo caso escrita em C++ poderia ser implementada da forma abaixo. As demais classes seria implementadas de forma semelhante, cada qual, com o comportamento que lhe é pertinente:

```
01:  class Círculo {
02:        int X, Y;
03:        int Raio;
04:     public:
05:        Círculo (int InitX, int InitY, int InitRaio) { /* ... */ };
06:        void Mostra (void) { /* ... */ };
07:        void Oculta (void) { /* ... */ };
08:        void Expande (int ExpandeBy) { /* ... */ };
09:        void Contrai (int ContraiBy) { /* ... */ };
10:  };
```

A etapa seguinte a ser realizada é o processo de compilação. Em Aurora cada uma das classes é compilada separadamente, exatamente como no exemplo acima, sendo que o resultado produzido pelo processo de compilação, o metaobjeto, é inserido em uma biblioteca e passa a

partir deste instante a fazer parte do universo de metaespaços disponíveis no sistema.

A partir da existência de um metaobjeto, que representa a figura geométrica, objetos podem ser instanciados associados ao metaobjeto e seus métodos ativados. Note que este processo de execução pode ser realizado diretamente através da interface do sistema. Por exemplo, para construir um círculo bastaria ser executado:

```
& CC objeto( círculo(10,12,7) )    // cria o objeto círculo
& CC.Expande( 15 )                 // aumenta o raio do círculo
```

"Objeto" representa o mecanismo de Aurora para instanciação de objetos, a partir de metaobjetos conhecidos pelo sistema. Supondo desejado um objeto de controle através do qual, por algum processo de escolha, o algoritmo decide qual das figuras geométricas deve ser construída, isto é, deseja-se utilizar as classes no interior de outro objeto. Para ter acesso aos metaobjetos basta o usuário incluir no objeto de controle a diretiva #include (para C++). A instanciação e a ativação do objeto mantém a mesma sistemática, ou seja:

```
#include <ObjetoAurora.h>
...
CC objeto( círculo(10,12,7) )
CC.Expande( 15 )
...
```

O leitor deve observar que a implementação do problema através do processo tradicional implica na necessidade do sistema conter o código que representa todas as classes previstas, mesmo que, somente uma das figuras geométricas seja utilizada ao longo da execução do sistema.

Note também que o enfoque de compilação e execução implementado por Aurora não é excludente, ou seja, caso o usuário eventualmente, ou sempre, desejar utilizar a forma monolítica para construção de sistemas, Aurora não impõe nenhuma restrição.

5 Considerações e Trabalhos Futuros

Este artigo apresentou o enfoque Aurora para compilação e execução de sistemas orientados a objetos, onde objetos, ou grupos de objetos, podem ser isoladamente instanciados em função de classes de objetos previamente conhecidas, ou dinamicamente configuradas, pelo sistema.

Neste aspecto Aurora difere da maioria dos sistemas, primeiro porque estes estão apoiados no modelo de programas compilados e *link* editados, segundo porque para a maioria dos sistemas uma classe é um padrão estático e imutável para os objetos. Em Aurora a construção hierárquica das classes pode ser transferida para o tempo de execução, permitindo que uma

classe herde efetivamente novas propriedades de forma dinâmica. Neste particular, até onde vai nosso conhecimento, Aurora é o primeiro sistema multilinguagem a permitir a transferência desta tarefa para o tempo de execução.

Aurora utiliza o modelo estrutural de Apertos[10] baseados na separação de objetos/metaobjeto visto que este facilita a implementação de modelos uniformes, ou seja, sistema e aplicações são construídos baseados na mesma abstração. Tal facilidade não é encontrada na maioria dos outros modelos estruturais, por exemplo a definição da abstração básica dos sistema baseados em *kernel* é definida por ele. Exemplos desta restrição podem ser encontradas em sistemas como, Amoeba[6], Chorus[5] e Choices[2], onde o kernel define níveis de abstração distintos, isto é, o kernel define a interface para o objeto da aplicação.

Uma característica que distingue Aurora é ser puro, isto é, suportar exclusivamente linguagens orientadas a objetos. Além de permitir um tratamento uniforme tanto para o sistema como para as aplicações do usuário, a adoção deste modelo facilita a integração entre o sistema operacional e as linguagens de programação, apesar de originalmente desenvolvidos de forma independente.

Note ainda, que a interface alcançada por Aurora reflete um ambiente exatamente como preconizado pelo paradigma de orientação a objetos, de que o processamento deve ser representado por um conjunto de mensagens fluindo entre objetos executando de forma paralela, enquanto a execução de sistemas restringe-se a criação de objetos e ao envio de mensagens.

No estágio atual, Aurora possui um protótipo completo da hierarquia de metaespaços responsáveis pelo suporte ao modelo estrutural validado, e apresentando grau de estabilidade considerável. A implementação de um protótipo como meta inicial do modelo Aurora, contribuiu rapidamente na avaliação e validação do modelo, permitindo dimensionar na prática as decisões de projeto adotadas. Tal protótipo está implementado em uma estação de trabalho SPARC2 em C++. Os resultados já alcançados, além auxiliarem no (re)direcionamento do projeto, encorajam sua implementação definitiva.

A máquina alvo de Aurora é uma arquitetura multiprocessadora baseada em Transputers disponível no curso de pós-graduação em ciência da computação da Ufrgs. O projeto Aurora por sua vez, se constitue no carro chefe da pesquisa realizada pelo autor para obtenção do grau de doutor, cujo enfoque principal é o problema do gerenciamento de objetos sobre arquiteturas multiprocessadas.

Referências

[1] AGHA, GUL. "Concurrent Object-Oriented Programming". Communications of the ACM, vol.33, No.9, pp. 125-141, 1990.

[2] CAMPBELL,R. et al. "Choices (Class Hierarchical Open Interface for Custom Embbedded Systems)". Operating Systems Review, vol 21, No 3, Jul 1987.

[3] CHAMBERS,C. "The Design and Implementation of the SELF Compiler, an Optimizing Compiler for Object-Oriented Programming Languages", Phd Thesis, Stanford University, Mar 1992.

[4] GOLDBERG,A. "Smalltalk-80, The Interactive Programming Environment". Addison Wesley, 1984.

[5] HERRMANN,F. et al. "Chorus distributer operating system". Computing Systems, vol 1(4), 305-367, 1988.

[6] MULLENDER,S.J. et al. "Amoeba - A Distributed Operation System for the 1990s". IEEE Computers, vol 23, 44-53, May 1990.

[7] NELSON, MICHAEL L. "Concurrency & Object-Oriented Programming", ACM Sigplan Notices, Vol.26, No.10, pp. 63-72, Out 1991.

[8] NICOL,J.R. et al. "Cosmos: an architecture for a distributed programming environment". Computer Communications, vol 12, No 3, 147-157, June 1989.

[9] TOMLINSON, CHRIS; SCHEEVEL, MARK. "Concurrent Object-Oriented Programming Languages", Object-Oriented Concepts, Databases, and Applications, ed. Won Kim and Frederick H. Lochovsky, pp.79-124, 1989.

[10] YOKOTE,YASUHITO: "The Apertos Reflective Operating System: The Concept and Its Implementation", Tecnical Report, Sony Computer Science laboratory Inc., 1992.

[11] YANEZAWA;AKINORI; TOKORO, MARIO. "Object-Oriented Concurrent Programming: An Introduction", MIT Press Series in Computer Systems, Massachusetts, 1988.

[12] ZANCANELLA,L.C. and NAVAUX,P.O.A. "AURORA: Um sistema Operacional orientado a objetos para arquiteturas multiprocessadoras". a ser publicado nos Anais do V SBAC-PAD Florianópolis, Sep 1993.

[13] ZANCANELLA,L.C. and NAVAUX,P.O.A. "Herança Dinâmica em AURORA". a ser publicado nos Anais do XX SEMISH, Florianópolis, Sep 1993.