AURORA:
UM SISTEMA OPERACIONAL ORIENTADO A OBJETOS para ARQUITETURAS MULTIPROCESSADORAS

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

Pós-Graduação em Ciência da Computação
Instituto de Informática - UFRGS/RS
Av. Bento Gonçalves, 9500 - Bloco *IV*
91.501-970 - Porto Alegre/RS - Brasil
Tel.: (051)336-8399 (ramal 6804)
email: zancanel@inf.ufrgs.br

Resumo

Este artigo apresenta Aurora, um sistema operacional orientado a objetos. Aurora foi projetado para combinar os benefícios do modelo orientado a objetos com as arquiteturas multiprocessadoras. O conceito de migração de objetos é utilizado para implementar os serviços do sistema operacional e distribuir a carga do sistema entre os processadores. Aurora suporta o conceito de herança dinâmica e trata todos os objetos de maneira uniforme, independente de localização.

Palavras-chave: sistemas operacionais, orientação a objetos, multiprocessadores

Abstract

This paper describes Aurora, an object-oriented operating system. Aurora has been proposed to combine the advantages of object oriented model and multiprocessors. Object migration is used to implement operating systems services and load balancing. Aurora suports dynamic inheritance and manipulates all objects in uniform way.

Keywords: operating systems, object-oriented, multiprocessor

[1]Mestre em Eng. de Sistemas e Computação, (COPPE/UFRJ,1983); Doutorando do CPGCC/UFRGS; Professor Afastado da UFSC/SC; Sistemas Operacionais, Multiprocessadores, Orientação a Objetos

[2]Professor UFRGS/CPGCC; Dr. Eng. em Informática (Instituto Nacional Politécnico de Grenoble, França,1979); Arquitetura de Computadores, Processamento Paralelo, Avaliação de Desempenho

1 *Introdução*

Apesar de não claramente perceptível, os últimos anos tem marcado o surgimento de uma nova geração de sistemas operacionais[YOK 92], mais dinâmicos, mais flexíveis e capazes de suportar de forma transparente a presença de processamento cooperativo, distribuído ou não, heterogêneo ou não. Vários fatores podem ser identificados como responsáveis por esta nova geração, dentre os quais a evolução do hardware, da tecnologia de comunicações, da engenharia de software e a necessidade de um incremento qualitativo, principalmente em ambientes de programação e interfaces[NIC 89].

A intensificação do uso de processamento paralelo, uma das características da evolução do hardware, tem sido alcançada tanto pela execução simultânea de diversas atividades, quanto pela realização antecipada de algumas atividades. São idéias largamente empregadas nas arquiteturas atuais, conceitos como: superposição das fases de execução da instrução; busca antecipada de instruções; multiplas unidades funcionais; processamento vetorial; estágios de execução e multiprocessamento.

Viabilizados recentemente, com o advento do VLSI, as arquiteturas multiprocessadoras não somente conquistaram espaço nas aplicações científicas, como também tornaram-se disponíveis comercialmente. Entretanto, tais sistemas não apresentam um incremento na sua capacidade de processamento linear com o número de processadores, reduzindo a taxa de utilização dos N processadores para uma razão entre $(\log_2 N)$ e $(N/\ln N)$[HWA 84]. Existem restrições de natureza física para esta perda, contudo, uma das principais razões reside no fato de que tais sistemas são construídos para aplicações genéricas, por conseguinte de considerável complexidade.

Por outro lado a evolução da engenharia de software, em particular em direção ao modelo de objetos, introduziu conceitos e abstrações, que encapsulam naturalmente dentro do próprio modelo de objetos, muitos dos problemas envolvidos no projeto de sistemas operacionais, tais como: identificação, proteção, atomicidade e sincronização.

Outro aspecto encorajador e que parece confirmar a adequabilidade do modelo de objetos para construção de sistemas operacionais, é a possibilidade dos recursos do sistema e as aplicações do usuário, serem modelados em termos da mesma abstração, o que introduz aos sistemas operacionais uma habilidade adicional para manipular o comportamento transparente e dinâmico dos sistemas, visto que recursos, serviços e o próprio sistema podem ser modelados de forma abstrata.

Assim se pensarmos que programas concorrentes, tais como sistemas operacionais, ao serem construídos são representados como uma coleção de objetos concorrentemente executáveis, nos induziremos a pensar que: o paradigma de objetos e a exploração simultânea do paralelismo, representam uma combinação poderosa tanto para o desenvolvimento como para a execução de programas concorrentes.

Por outro lado, parece claro que a obtenção de um bom ambiente de programação e

execução orientado a objetos, somente pode ser conseguido através de uma arquitetura adequada, visto que tais ambientes, pela sua natureza, devem incorporar em tempo de execução; criação dinâmica de objetos, acoplamento dinâmico, buferização de mensagens, pesquisa a métodos e coleta de lixo.

Entretanto, a utilização de arquiteturas adequadas não é realística, consequentemente apesar de não serem arquiteturas[INM 90] particularmente adequadas ao modelo de objetos, a proliferação das arquiteturas multiprocessadoras representa não somente a oportunidade, mas o s' porte fundamental, para a modelagem do ambiente programação e execução adequado a orientação a objetos.

Deste modo, o surgimento de Aurora, representa um passo em direção a busca de um modelo adequado ao paradigma de objetos, capaz de proporcionar através da exploração eficiente das arquiteturas multiprocessadoras, um suporte de software para a modelagem de um ambiente apropriado para execução de objetos.

2 *O Sistema Aurora*

O surgimento de Aurora foi inspirado na premissa do modelo de objetos, que se propõe a modelar objetos que no mundo real são naturalmente concorrentes (e distribuídos) e cujo processamento das informações em um ambiente baseado em objetos, pode ser simplificadamente representado como, um conjunto de mensagens fluindo entre objetos executando de forma paralela.

Deste modo, Aurora foi projetado para explorar o paralelismo, tanto a nível do sistema como a nível da aplicação, visando suportar o desenvolvimento de aplicação modeladas em termos de objetos de forma paralela. Em vista disso, Aurora está baseado no modelo de programação concorrente orientado a objetos[AGH 90, KAF 89, NEL 91, TOM 89, YON 88], onde 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, de modo a viabilizar tratamento uniforme aos objetos, independente dos mesmos estarem implementando aplicações do usuário ou serviços do sistema. Tais abstrações incorporam em tempo de execução, suporte às características intrínsecas da execução orientada a objetos, tais como: criação e acoplamento dinâmica de objetos, gerenciamento de mensagens, pesquisas a métodos e coleta de lixo.

Um problema a ser enfretado, é que a atual tecnologia de desenvolvimento de sistemas operacionais, é baseada em modelos entruturais voltados ao suporte de processos e não possuem habilidade para manter e gerenciar objetos eficientemente[CHI 91], entidade de

granularidade mais leve que processos.

Aurora utiliza o modelo estrutural de Apertos[YOK 92], que implementa a idéia de que o nível de objetos e o nível de abstração das linguagens possam ser separadamente descritos e implementados dentro da mesma estrutura. Deste modo, Aurora, utiliza o conceito de separação entre objetos e meta-objetos, onde o objeto, a exemplo de Apertos, representa o estado do objeto (um depositório de dados), enquanto o meta- objeto define, não somente a semântica de seu comportamento, como em Apertos, mas a abstração da classe.

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 ao objeto um conjunto de abstrações que definem a semântica do comportamento dos objetos pertencentes a este meta-espaço.

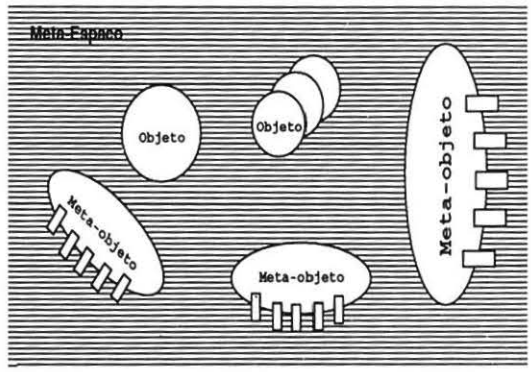

Figura 1: Visão do modelo estrutural de Aurora

Deve-se observar, que serviços do sistema operacional e aplicações do usuário são implementados através de meta-objetos, cada um dos quais fazendo parte de algum meta-espaço. Deste modo, o modelo computacional de Aurora é composto por um conjunto de meta-espaços, distribuídos entre os processadores e cooperando na relização de tarefas.

A utilização do conceito de separação entre; objetos e meta-objetos, acrescida da habilidade dos meta-objetos representarem a abstração da classe, acrescenta a Aurora, a capacidade para implementar o conceito de herança dinâmica de classe[ZAN 93a], ou seja, permite que a construção hierárquica das classes possa ser realizada durante a execução do sistema 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.

Deve-se observar que a introdução do conceito de herança dinâmica, possibilita uma total integração entre o ambiente de programação e o sistema operacional e trás consigo um novo conceito de compilação e execução[ZAN 93b], onde a função básica do compilador passa agora a ser de gerar meta-objetos, que serão inseridos em uma bliblioteca. Os meta-objetos existentes nesta biblioteca podem ser referenciados por objetos do usuários, ou do sistema, e serão utilizados pelo sistema operacional quando da instanciação de objetos, ou quando da ativação de métodos externos ao meta-objeto instanciado.

A figura 2 apresenta uma visão do ambiente Aurora e da interface projetada, onde o sistema é apresentado como uma coleção de classes (quadros xadrez pelas classes Q e C++) e meta-objetos (quadro no canto esquerdo), permitindo ao usuário desenvolver aplicações baseadas nas classes existentes e/ou interagir diretamente com meta-objetos através da ativação dos mesmos, seja a nível executável seja a nível da linguagem de interação.

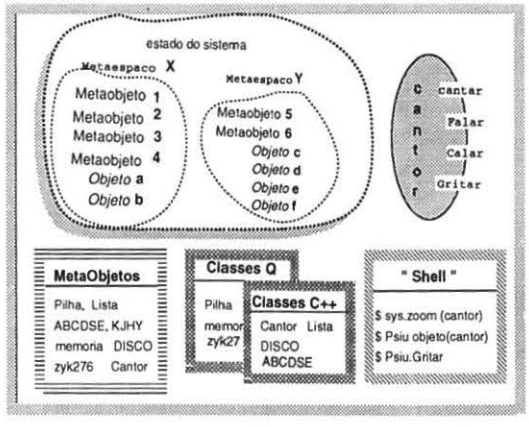

Figura 2: Visão Simplificada do ambiente e da Interface de Aurora

A interface mostrada no quadro zebrado, salienta a interação do usuário diretamente com um objeto do sistema (sys.zoom(cantor)) e criando sua própria aplicação, através da instancição de um objeto (Psiu objeto(cantor)) a partir um meta-objeto conhecido e ativando este objeto através da ativação de um de seus métodos (Psiu.Gritar). A figura pontilhada apresenta uma possível visualização do estados do sistema em um determinado instante, ou seja, os meta-espaços existentes.

O mecanismo básico para construção do sistema operacional 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. Da mesma forma, um objeto pode migrar para um meta-espaço que implementa uma impressora quando é para ser impresso.

Conforme salientado, um dos propósitos de Aurora é contribuir para a consolidação da atual tecnologia relacionada ao problema de gerenciamento de objetos. Deve-se notar, que o fato de Aurora ser modelado de forma abstrata em todos os níveis, acrescenta ao sistema a flexibilidade de permitir a troca de políticas sem a necessidade de modificação dos mecanismos básicos. Tal facilidade contribuirá na busca de um modelo adequado, capaz de permitir que objetos sejam mantidos, gerenciados e usados de forma eficiente.

Deste modo, nesta primeira implementação, características intrinsecamente relacionadas com o aspecto de gerenciamento de objetos, tais como: estrutura dos objetos, controle das ativações, sincronização, segurança e confiabilidade, apresentam soluções clássicas. A busca de soluções particulares será intencificada tão logo o sistema atual apresentar grau de estabilidade considerável. A tarefa de gerenciamento da interação entre objetos, entretanto apresenta peculiaridades próprias, visto que Aurora suporta o conceito de herança dinâmica. Uma descrição pormenorizada de tais características é encontrada em [ZAN 93a].

Assim sendo, quanto a estrutura dos objetos, Aurora apresenta o modelo passivo e o suporte a livre granularidade. A utilização do modelo de objetos passivos parece ser conveniente neste primeiro momento, visto que Aurora trata todos os objetos do sistema de maneira uniforme, quer estejam na memória local, na memória associada a outro processador ou armazenados em um dispositivo secundário. A facilidade para suporte a livre granularidade é oriunda do fato de que o modelo proposto é um ambiente completamente uniforme e consistente para todas as entidades, desprezando o fato do objeto ser caracterizado como grande ou pequeno.

O controle das ativações sobre os objetos é realizada de forma individualizada dentro de cada meta-espaço, de modo que, cada meta-espaço garanta que múltiplas ativações sobre os objetos que pertencem ao meta-espaço, sejam executadas de maneira que, o efeito resultante corresponda ao mesmo da execução seqüêncial. Deve-se observar ainda que a ativação de um método é executada de forma atômica, ainda que produzindo várias invocações conexas e afetando mútiplos objetos. A atomicidade na ativação de um método, significa que a ativação é executada com sucesso ou então não tem efeito.

A garantia de que múltiplas ativações sobre o mesmo objeto não conflitam entre si, é suportada através de mecanismos de sincronização pessimistas, quando a ativação afeta múltiplos objetos. Mecanismos de sincronizacão otimistas são usados quando um único objeto é afetado. Nenhuma preocupação com segurança e confiabilidade são adotadas nesta primeira proposta.

Outra característica de Aurora é a transparência ao escalonamento dos processadores. Visto que um dos objetivos de Aurora é a busca de alto desempenho na utilização das arquiteturas multiprocessadoras, deve ser adotada, um cuidadoso critério na distribuição

dos objetos, de modo a minimizar referências remotas, visto que tais referências são em geral mais dispendiosas que as referências locais e, embora 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.

A caracterização de Aurora como sendo um sistema orientado a objetos, é baseada na definição de Wegner[WEG 89] para linguagens de programação, de que uma linguagem somente é orientada a objetos, caso esteja presente algum dos conceito entre herança e delegação. Neste sentido, pode-se dizer que Aurora é um sistema orientado a objetos, visto que o mesmo suporta herança dinâmica tanto sobre classes como sobre objetos.

3 *Implementação de Aurora*

Note que Aurora é modelado como um ambiente uniforme em todos os níveis, ou seja, aplicações do usuário, serviços do sistema operacional, dispositivos (tais como discos ou redes de conexões) e mesmo a memória são definidos como objetos ou coleções de objetos. A presença de meta-espaços permite que cada objeto tenha seu próprio ambiente de execução, proporcionando ao objeto a habilidade para alterar seu comportamento durante a execução do sistema.

Desta forma, o ambiente pode ser visto como definido dentro de uma hierarquia de meta-espaços, onde no topo da hierarquia esta implementado o suporte ao modelo estrutural, a partir do qual, o sistema operacional, utilitários e aplicações do usuário são construídos. Neste nível estão também implementadas as características intrínsecas a execução orientada a objetos. A implementação deste suporte é realizada através de meta- espaços que implementam basicamente mecanismos para:

- Gerenciamento de meta-espaços.
- Suporte à ativações.
- Suporte à herança dinâmica.
- Suporte ao multiprocessamento

A figura 3 apresenta uma visão simplificada do modelo implementado, onde o suporte ao modelo estrutural constitue no topo da hierarquia, a partir do qual sucessivos meta-espaços implementando mecanismos, serviços do sistema e aplicações do usuário, compartilham facilidades comuns, como por exemplo migração de objetos. Deve-se observar que alguns meta-espaços não possuem objetos associados. Esta particularidade ocorre

principalmente nos meta-espaços que implementam basicamente serviços do sistema operacional.

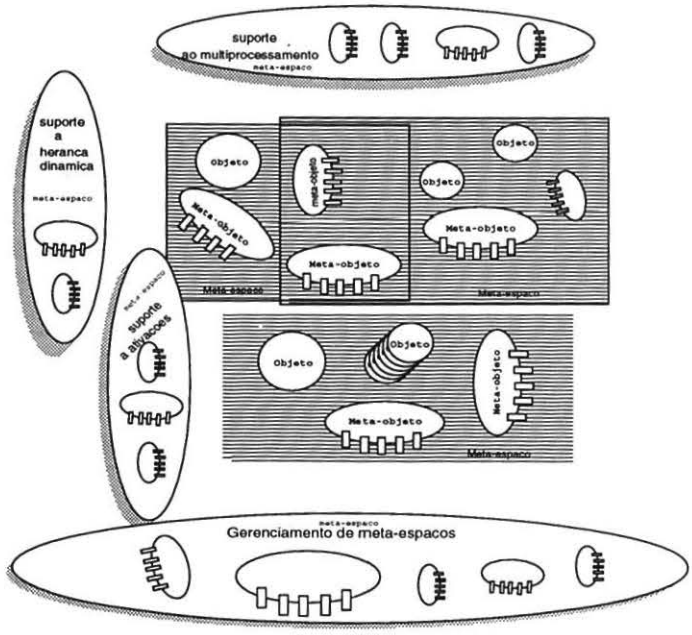

Figura 3: Visão Simplificada da Implementação de Aurora

A tabela 1 descreve o conjunto de facilidades providas para gerenciamento de meta-espaços. Tais facilidades são utilizados por exemplo quando da instanciação de um objeto e, após determinada a necessidade de criação de um novo meta-espaço, para suportar o objeto. Note que os mecanismos responsáveis pelo suporte ao multiprocessamento são ativados, a fim de determinar sobre qual processador o novo meta-espaço será criado.

Tabela 1: Gerenciamento de meta-espaços

Primitiva	Função
NewMetaSpace	Criar um objeto descritor de meta-espaço
KillMetaSpace	Deletar o objeto descritor de meta-espaço
NewMetaObject	Acrescenta um novo meta-objeto ao meta-espaço
NewObject	Implementa criação dinâmica de objetos
ShowMetaSpace	Examinar o conteúdo do meta-espaço
CheckMetaSpace	Utilizado para localizar objetos e/ou meta-objeto

Deve-se notar, que todo meta-espaço contém internamente meta-objetos ditos *standard*, associados em tempo de criação do meta-espaço e que implementam funções de auto-gerenciamento do meta-espaço, tais como escalonamento interno para controlar a execução das ativações realizadas sobre os objetos. Suporte a migração de objetos também é implementado através de meta-objetos *standard* de modo que um objeto pode auto-migrar para outro meta-espaço quando desejar.

A referência a um objeto através da ativação de um de seus métodos, é suportada pelo suporte ao controle de ativações, cujas facilidades estão descritas na tabela 2. Observe que este meta-espaço é responsável por outras tarefas além das tarefa básicas de comunicação sincrona e assincrona suportadas na implementação corrente.

Deve-se notar que existem situações distintas que podem ocorrer na ativação de um objeto. O caso mais simples é o meta-objeto já pertencer ao meta-espaço do objeto e neste caso a ativação corresponde a uma chamada local. Caso o meta-objeto pertença a outro meta-espaço, este pode ser local ou remoto. No primeiro caso, o meta-objeto passa a ser compartilhado entre os meta-espaços e a ativação procede como no caso anterior. No caso do meta-espaço estar localizado em um processador remoto, a mensagem é transformada em uma chamada remota. No caso do meta-objeto não ser encontrado, o mesmo é acrescentado ao meta-espaço a que o objeto pertence, e a ativação torna-se uma chamada local. Note que a responsabilidade desta tarefa é do suporte a herança dinâmica, cuja descrição pode ser encontrada na referência.

Tabela 2: Suporte a ativações

Primitiva	Função
ProcCall	Implementa uma ativação a um método do objeto de destino, semelhantemente a uma chamada de procedimento.
MsgSend	Implementa uma ativação a um método do objeto de destino, semelhantemente ao envio de uma mensagem assincrona.
FindObject	Localiza o objeto de destino.
Binding	Implementa acoplamento dinâmico de objetos

A arquitetura alvo de Aurora, nesta primeira implementação, é baseada em Transputers[INM 89]. Entretanto, deve-se notar que o modelo conceitual de Aurora é independente de arquitetura, de modo que a utilização deste hardware em particular é circunstancial. Desta forma a única suposição de Aurora, a respeito do hardware, é que ele consiste de Transputers interconectados via links, não sendo assumido nenhuma topologia particular.

A arquitetura multiprocessada empregada consiste de uma rede de Transputers interconectados através de uma placa IMS B008[INM 90]. A configuração atual contém quatro *TRAnsputer Module (TRAM)* disponíveis para execuções paralelas, conectados *pipeline*. O estabelecimento de uma rede de interconexão particular entre os *TRAMs* pode ser realizada por software através de um transputer dedicado para este fim na placa IMS B008.

Uma característica do hardware é que toda comunicação entre os *TRAMs* e entre *TRAMs* e a interface PC bus (hospedeiro) é realizada via links.

A figura 4 apresenta a configuração atual da placa IMS B008, onde os *links* físicos são utilizados para estabelecer a conexão básica (*pipeline*). Interconexões lógicas podem ser estabelecidas para definição de topologias particulares, no exemplo mostrado o *link lógico* é utilizado para configurar uma rede em anel.

Cada *TRAM* dispõe de 1 *megabyte* de memória acoplado, sendo que o hardware descrito não contém qualquer forma de memória compartilhada. Deste modo Aurora não apresenta nenhum serviço centralizado sobre o qual o sistema completo atue. A adoção desta política acrescenta ao sistema certa confiabilidade, visto que a falha ocasional de um processador não necessariamente causará um colapso total no sistema.

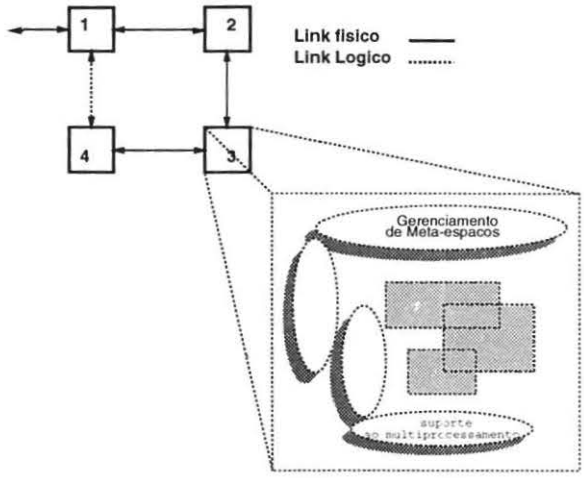

Figura 4: Rede de Transputers

Para obter esta natureza descentralizada, totalmente transparente tanto para o usuário como para o sistema em execução, Aurora distribui em cada processador uma cópia dos meta-espaços que implementam o suporte ao modelo estrutural (figura 4), de modo que: criação dinâmica de objetos, acoplamento dinâmico, gerenciamento de mensagens, pesquisas a métodos e coleta de lixo, características intrínsecas da execução orientada a objetos, possam ser realizadas localmente ao processador.

A tabela 3 descreve o conjunto de facilidades providas para suporte a multiprocessameto. Tais facilidades são utilizados para controlar o balanceamento de carga sobre o sistema, de modo que ao ser criado um novo meta-espaço, os mecanismos providos neste modulo são ativados a fim de determinar sobre qual dos processadores o mesmo deve

ser criado. Da mesma forma, sempre que um meta-espaço é destruído, as condições de balanceamento de carga são analisadas, podendo ocorrer migração de meta-espaços para redistribuição da carga sobre os processadores.

Tabela 3: Suporte ao multiprocessamento

Primitiva	Função
CheckBalancing	Implementa uma análise rápida do balanceamento
LoadBalancing	Determina um processador para criação de um meta-espaço
NewBalancing	Realiza uma análise aprofundada da carga no sistema, de modo a estabelecer um novo balanceamento de carga

No estágio atual, Aurora possui um protótipo completo da hierarquia de meta-espaços 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 alcançados, além de auxiliarem no (re)direcionamento do projeto, encorajaram sua implementação definitiva, sobre o hardware descrito.

4 *Conclusões*

Este artigo apresentou Aurora, um sistema operacional orientado a objetos para arquiteturas multiprocessadoras. Semelhantemente a outras propostas, o objeto é a unica entidade constituinte do sistema. Entretanto, um objeto Aurora não é somente um conjunto de propriedades e semânticas, visto que a semântica e as propriedades do objetos são alteradas enquanto o objeto está executando e quando ele evolue ou muda de meta-espaços.

O modelo proposto torna a exploração de arquiteturas multiprocessadoras mais flexível, visto que o suporte a migração de objetos faz parte do modelo conceitual. Esta mobilidade natural atribuída aos objetos, permite que o sistema gerencie a execução paralela entre os mesmos com um baixo custo, já que os objetos são entidades de granularidade mais leve que processos.

Outro aspecto saliente da vantagem da utilização de uma arquitetura multiprocessada na implementação de sistemas orientados a objetos é a possibilidade do sistema tratar em paralelo com a aplicação, problemas críticos e catacterísticos da execução orientada a objetos, tais como: criação dinâmica de objetos, acoplamento dinâmico, gerenciamento de mensagens, pesquisas a métodos e coleta de lixo.

Aurora implementa um novo enfoque 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 durante a execução do sistema.

Uma característica singular de Aurora é, tranferir para tempo de execução, a construção hierárquica das classes, 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 tempo de execução.

Outra propriedade que distingue Aurora é ser puro, isto é, suportar exclusivamente linguagens orientadas a objetos. Além de permitir um tratamento uniforme tanto para o sistemas como para as aplicações do usuário, a adoção deste modelo facilita a integração entre o sistema operacional e às linguagens de programação, apesar de originalmente desenvolvidos de forma independente. Tal integração representa uma base fundamental na busca de ambientes de programação e interfaces qualificados.

Referências

[AGH 90] AGHA, GUL. "Concurrent Object-Oriented Programming". Communications of the ACM, vol.33, No.9, pp. 125-141, 1990.

[CHI 91] CHIN,R.S.,CHANSON,S.T.,"Distributed, Object-Based Programming Systems". IEEE Software, ACM Computing Surveys, Vol 23, No 1, Mach 1991.

[INM 89] INMOS. "The Transputer Databook", INMOS Limited, 1989.

[INM 90] INMOS. "Iodule Motherboard Architecture: IMS B008", Reference Manual. INMOS Limited, 1990.

[HWA 84] HWANG K., BRIGGS F.A. "Computer Architecture and Parallel Processing", McGraw-Hill Book Company, 1987.

[KAF 89] KAFURA, D.G; LEE, K.H. "Inheritance in Actor Based Concurrent Object-Oriented Languages". The Computer Journal, Vol. 32 No. 4, pp. 297-304, Ago 1989.

[NEL 91] NELSON, MICHAEL L. "Concurrency & Object-Oriented Programming", ACM Sigplan Notices, Vol.26, No.10, pp. 63-72, Out 1991.

[NIC 89] NICOL,J.R. et al. "Cosmos: an architecture for a distributed programming environment". Computer Communications, vol 12, No 3, 147-157, June 1989.

[TOM 89] 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.

[YOK 92] YOKOTE,YASUHITO: "The Apertos Reflective Operating System: The Concept and Its Implementation", Tecnical Report, Sony Computer Science Inc., 1992.

[YON 88] YANEZAWA;AKINORI; TOKORO, MARIO. "Object-Oriented Concurrent Programming: An Introduction", MIT Press Series in Computer Systems, Massachusetts, 1988.

[ZAN 93a] ZANCANELLA,L.C. and NAVAUX,P.O.A. "Herança Dinâmica em AURORA", a ser publicado no Anais do XX SEMISH, Florianópolis, Sep 1993.

[ZAN 93b] ZANCANELLA,L.C. and NAVAUX,P.O.A. "Os processos de compilação e execução em AURORA", submetido ao VII Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, Oct 1993.

[WEG 89] WEGNER,P. "Learning the Language", BYTE, pp.245-254, Mar 1989.