1
Introdução

Normalmente, quando começamos a aprender a programar, nos deparamos com estruturas de código que seguem o conceito de programação seqüencial. Rapidamente o programador acredita que em seus programas sempre existirá um conceito de "programa principal", que toma conta da aplicação e é responsável por chamar funções e subrotinas definidas, aguardar seus resultados e a partir desses tomar decisões que definirão o fluxo do programa. É normal que para simplificar o aprendizado, essas chamadas sejam efetuadas de maneira síncrona. Quando introduzimos o paradigma de programação orientada a eventos, vários desses preceitos devem ser deixados de lado. O que passamos a ter é um sistema onde eventos são detectados e passados para os programas que estão aguardando sinalizações desses eventos. Uma vez processados, a aplicação devolve o controle para o sistema operacional, que se encarrega de aguardar os próximos eventos. Essa inversão do controle da aplicação é conhecida como *The Hollywood Principle – Don't call us, we'll call you* [36]

Desde 1980, essa inversão do controle da aplicação, vem se tornando cada vez mais evidente, como por exemplo em aplicações gráfico-interativas, onde a interface gráfica com o usuário (*Graphical User Interface - GUI*) dispõe apenas de mecanismos de reação às ações deflagradas pelo usuário através dos mecanismos de entrada. Além disso, diversos sistemas distribuídos que têm na troca de mensagens um mecanismo de comunicação e disparo de ações; essas mensagens podem ser interpretadas como eventos de entrada para o sistema que as recebe, que deve reagir ao seu conteúdo executando a ação esperada pelo remetente. Também podemos citar ambientes que recentemente vêm ganhando força como dispositivos computacionais como os celulares, que com a atual revolução tecnológica, trazem telas maiores e coloridas e a possibilidade de se executar programas em processadores comparáveis aos primeiros PCs. Esses dispositivos funcionam de maneira exclusivamente orientada a eventos, respondendo a requisições da rede, provenientes dos sistemas de antenas, e dos usuários, através das

interfaces gráficas.

Nos sistemas orientados a eventos, a necessidade de estar sempre preparado para o processamento de uma requisição traz consigo restrições de desempenho e controle do fluxo de dados que muitas vezes fogem aos métodos triviais de programação. A pouca familiaridade com os paradigmas de desenvolvimento orientado a eventos é muitas vezes contornada com o uso de políticas preemptivas para o uso de recursos do sistema. A preempção permite um gerenciamento transparente para o desenvolvedor, porém, além de depender do suporte oferecido pelo sistema operacional, acarreta vários efeitos colaterais, que podem ser observados no aumento da dificuldade de depuração em sistemas complexos, ocasionado por métodos com fluxo não linear de dados, e na dificuldade no controle dos dados compartilhados, além do custo em termos de desempenho gerado pelas estruturas utilizadas na manutenção desse controle de acesso.

1.1 Objetivos

Frente às dificuldades apresentadas por essa inversão de controle nos sistemas orientados à eventos, esta dissertação propõe uma arquitetura para escalonadores de tratadores de eventos que busca minimizar os problemas encontrados durante o desenvolvimento de aplicações. Em nossa arquitetura, optamos por uma abordagem de escalonamento colaborativo, pois permite o encapsulamento do contexto de execução, eliminando a necessidade de estruturas de controle de acesso aos recursos compartilhados, criando um ambiente mais confortável para o desenvolvedor, se aproximando do cenário encontrado em aplicações com apenas uma linha de execução.

O escalonador é a estrutura responsável por encapsular o recebimento de eventos e tratá-los de acordo com sua prioridade, permitindo que a aplicação final se preocupe apenas em definir os procedimentos a serem executados no recebimento desses eventos, permitindo que eles sejam processados de maneira concorrente, garantindo assim maior escalabilidade para o sistema.

Essa arquitetura será explorada em dois ambientes bastante distintos. O primeiro deles propõe uma adaptação da máquina virtual Lua para o cenário da computação móvel celular usando a tecnologia BREW[6] - um cenário inerentemente assíncrono, com nenhum suporte a programas com múltiplas linhas de execução e pouco poder de processamento. O segundo trata de aplicações envolvendo chamadas remotas de métodos em um

middleware de comunicação CORBA[1]. Nesse cenário, o escalonador deverá coordenar a execução colaborativa de métodos de comunicação, permitindo o processamento concorrente de múltiplas requisições e suporte à reentrância de métodos sem comprometer o desempenho do *middleware*.

Durante a programação para dispositivos celulares, o desenvolvedor é exposto a um ambiente de programação bastante hostil, onde o fluxo de controle da aplicação é completamente controlado por eventos com diferentes prioridades. Nesse ambiente, grande parte da dificuldade no desenvolvimento de aplicações reside no correto gerenciamento desses eventos, provenientes de diversas fontes externas: o próprio usuário, notificações de chamadas de voz ou indicadores de nível de bateria. Além disso, a necessidade de estar sempre disponível para responder às requisições da rede obriga o programador a sempre se preocupar com a disponibilidade do sistema operacional, evitando executar tarefas síncronas que exijam muito tempo para serem concluídas. Nesse cenário, o escalonador de eventos proposto deve encapsular o controle de prioridades e implementar mecanismos de despacho de eventos de forma mais simples para o programador da aplicação final.

A necessidade de mecanismos eficientes para o tratamento de eventos também é uma realidade para a implementação de mecanismos de comunicação em sistemas distribuídos, como em chamadas remotas de métodos. Durante a fase inicial do desenvolvimento do *OiL* (*ORB in Lua*)[24], possuía-se um sistema capaz apenas de lidar de forma síncrona com apenas uma requisição por vez, eliminando a possibilidade de objetos de *callback* ou até mesmo de interação com o usuário. Mais uma vez surge a necessidade de gerenciamento dos eventos processados pela aplicação, dessa vez, em um ambiente originalmente de chamadas síncronas.

1.2 Abordagem do Problema

Nesta dissertação vamos apresentar modelos que permitam a aplicação do conceito de corrotinas como elemento fundamental na execução colaborativa de métodos em chamadas remotas ou em processamentos assíncronos não bloqueantes.

A linguagem Lua [27] vai ser utilizada como ambiente de desenvolvimento, visando aproveitar o alto grau de portabilidade oferecido pela linguagem. Lua implementa o conceito de corrotinas assimétricas, que envolve basicamente um método para (re)iniciar uma corrotina e outro método para suspendê-la, retornando o controle para o iniciador da corrotina [20].

Nosso objetivo é maximizar o aproveitamento do tempo disponível de processador e ao mesmo tempo manter um alto grau de disponibilidade dos serviços, possibilitando assim o tratamento reentrante de chamadas de métodos remotos e a execução colaborativa de métodos não bloqueantes. Isso tudo utilizando uma ferramenta leve, portável e simples.

O restante deste trabalho está organizado da forma descrita a seguir. No capítulo 2 é feita uma recapitulação de algumas arquiteturas de escalonadores de eventos, analisando suas vantagens e desvantagens dentro de modelos tradicionais de concorrência. No capítulo 3 é descrito o modelo proposto para um escalonador de eventos colaborativo usando corrotinas, visando simplificar o desenvolvimento de aplicações sem pôr em risco a portabilidade do sistema. O capítulo 4 apresenta o *middleware OiL* e o modelo de escalonador proposto para sua arquitetura. No capítulo 5 é feita uma análise do *LuaBREW*, um estudo de caso prático da implementação do modelo proposto em um ambiente completamente assíncrono. Por fim, no capítulo 6 são apresentadas algumas conclusões do trabalho e idéias para trabalhos futuros.