# 5
# Conclusão

As abstrações de programação para o desenvolvimento de aplicações que requerem interação assíncrona entre as partes de uma aplicação refletem, normalmente, o assincronismo da comunicação baseada na troca de mensagens não-bloqueantes. Cabe ao programador construir visões de nível mais alto, capazes de modelar apropriadamente a relação entre as partes de uma aplicação usando a lógica dos mecanismos de comunicação assíncrona. Ao longo do trabalho discutimos como essa forma de lidar com o assincronismo torna a tarefa do desenvolvedor de aplicações mais complexa e aprofundamos a discussão sobre como oferecer uma interface de programação mais apropriada, combinando operações de comunicação não-bloqueantes com um modelo de gerência cooperativa de tarefas.

Os sistemas operacionais trataram o assincronismo, necessário para lidar com vários dispositivos físicos independentes, introduzindo o modelo de multi-tarefa com preempção. Esse modelo foi estendido e chegou no nível de desenvolvimento de aplicações oferecendo a possibilidade de tratar as partes distintas de uma aplicação de forma independente, como contextos de execução separados que podem ser suspensos e retomados. Essa flexibilidade permite simplificar o tratamento do assincronismo, mas impõe um custo computacional excessivo em alguns contextos. Além disso, o modelo de multi-tarefa com preempção exige que o programador se preocupe em garantir a sincronização do acesso aos recursos compartilhados e torna a depuração das aplicações mais difícil.

Através do ambiente de execução e do conjunto de operações providos pelo ambiente LuaRPC, estudamos formas de combinar a comunicação assíncrona entre as partes de uma computação com modelos de gerência cooperativa, oferecendo uma interface de programação que permite construir diferentes visões para a interação entre os processos de uma aplicação. Usando as operações do ambiente LuaRPC o programador pode:

– requisitar a execução de um serviço com a opção de preservar o estado de execução corrente para retomá-lo quando a resposta estiver disponível (visão síncrona);

- divulgar um serviço que quando invocado será automaticamente tratado pela função de retorno (visão assíncrona);
- registrar interesse em um evento e usar a função de retorno como o tratador desse evento, executando essa função quando a notificação do evento for recebida (visão de eventos).

O programador pode explorar a forma de interação mais apropriada para cada tipo de problema — ou usar as diferentes formas de interação simultaneamente, dentro da solução do mesmo problema — mantendo a característica essencial da comunicação não-bloqueante no nível de troca de mensagens.

Linguagens com as características que enfatizamos e exploramos na implementação das operações providas pelo ambiente LuaRPC — funções como valores de primeira classe e *closure* — são normalmente linguagens interpretadas, que no contexto particular de aplicações paralelas representam perda de desempenho. Entretanto, como discutido em (Ururahy/02), se um modelo de programação dual é usado, no qual a linguagem interpretada coordena a interação entre as partes distintas da aplicação e uma linguagem compilada implementa a computação dessas partes, os resultados alcançados podem ser muito bons, especialmente considerando a flexibilidade que pode ser obtida nesse caso.

Experimentando a mesma combinação — comunicação assíncrona com gerência cooperativa de tarefas — dentro da arquitetura do sistema operacional TinyOS, mostramos que é possível oferecer ao programador uma interface de programação mais conveniente de usar. O custo computacional adicionado ao TinyOS com a estrutura de multi-tarefa cooperativa ficou dentro de um limite razoável e compatível com as restrições particulares do contexto de redes de sensores. Através da infra-estrutura provida com as operações e o escalonador de co-rotinas, tarefas naturalmente sequenciais em aplicações com sensores (como a leitura e transmissão de um valor), mas que requerem a interação assíncrona com dispositivos externos, podem ser implementadas dentro de um único procedimento da linguagem, simplificando o algoritmo da aplicação.

Usando co-rotinas como construção para permitir a estrutura de multi-tarefa, reduzimos o peso computacional comum das soluções que usam *threads* e gerência preemptiva. Explorando a idéia de um processamente mínimo associado às operações de interação, encapsulamos a transferência de controle entre os fluxos de execução dentro dessas operações. Assim, os pontos de troca de controle aparecem de forma explícita no código da aplicação, mas o programador não precisa lidar diretamente com a gerência das co-rotinas.

## 5.1 Sumário de contribuições

A primeira contribuição deste trabalho foi aprofundar a discussão sobre como lidar com contextos que requerem interações assíncronas entre as partes de uma aplicação sem sacrificar a tarefa do programador. O trabalho explorou a estratégia de combinar a arquitetura multi-tarefa com gerência cooperativa com o paradigma de comunicação baseado em eventos, favorecendo a distinção entre a visão do sistema e a visão do programador.

Investigamos o uso de co-rotinas no contexto de aplicações distribuídas e mostramos que a construção de co-rotinas é uma alternativa viável para oferecer uma infra-estrutura multi-tarefa com um custo computacional mais leve que o modelo tradicional baseado em *threads*. Combinando co-rotinas com um mecanismo de comunicação baseado em eventos, construímos operações de interação que oferecem ao programador uma visão síncrona — mais conveniente para modelar determinados problemas — sobre uma base assíncrona de comunicação. Essas operações permitem ainda explorar modelos distintos de programação dentro de uma mesma aplicação, uma facilidade conveniente para determinados contextos da computação distribuída.

O trabalho contribui ainda para a discussão sobre como prover interfaces de programação mais adequadas para o desenvolvimento de aplicações para redes de sensores. As limitações e restrições particulares desse contexto impõe a necessidade de investigar modelos alternativos de programação para não sobrecarregar a tarefa do programador.

## 5.2 Trabalhos futuros

A investigação realizada ao longo deste trabalho e os resultados alcançados mostram a necessidade de aprofundar o tratamento de alguns aspectos particulares e apontam novas direções de pesquisa.

O ambiente computacional LuaRPC simplificou o tratamento de questões relacionadas à comunicação via rede, como o endereçamento dos processos em máquinas distintas, para focar a investigação apenas na semântica das operações de interação. Uma linha de trabalho futuro é aprimorar o ambiente LuaRPC investigando alternativas apropriadas para a interface com a rede. Além disso, como o ambiente LuaRPC foi inteiramente implementado usando a linguagem Lua, outra linha de investigação é experimentar o uso das operações providas em um modelo de programação dual, onde as operações providas pelo LuaRPC são usadas para coordenar a interação entre as partes da aplicação e a computação dessas partes é implementada por uma linguagem compilada.

Outra linha de investigação dentro do ambiente implementado por LuaRPC, é explorar com mais profundidade a mistura de modelos de programação distintos dentro de uma mesma aplicação, por exemplo, os modelos dirigido a eventos e os modelos baseado em invocaç oes de métodos remotos.

Para alterar a arquitetura do TinyOS trabalhamos com a versão 1.x do sistema. Essa versão implementa o escalonador de tarefas como parte do núcleo do sistema operacional. Para incluir o escalonador de co-rotinas dentro do TinyOS foi preciso alterar diretamente o núcleo do sistema operacional re-compilá-lo. Desse modo, mesmo as aplicações que não precisam da infra-estrutura com co-rotinas para simplificar a programação, obrigatoriamente arcam com o custo computacional adicionado pelo escalonador de co-rotinas. O caso especial de aplicações com requisitos de tempo real, para as quais o custo adicionado com a infra-estrutura proposta pode não se justificar, é outro exemplo no qual a alteração do núcleo do sistema operacional não se mostra como uma opção adequada. Uma alternativa mais apropriada seria implementar a infra-estrutura com co-rotinas como um componente a parte, o qual poderia ser incluído nas aplicações como os demais componentes TinyOS. Uma das principais inovações propostas pela versão 2.0 do TinyOS é permitir que o escalonador de tarefas seja implementado como um componente adicional, que desse modo pode ser mais facilmente alterado para atender às necessidades específicas de diferentes aplicações. Um trabalho futuro é explorar essa facilidade provida com a versão 2.0 do TinyOS para definir as operações com co-rotinas em uma interface e tratar o escalonador de co-rotinas como um componente que implementa essa interface. Nessa linha de trabalho, queremos também otimizar nossa implementação inicial do escalonador e da própria contrução de co-rotinas.

Outra linha de trabalho consiste em aprofundar o estudo sobre a aplicabilidade das facilidades de programação providas com as interfaces de alta nível, considerando os diferentes tipos de aplicações para as redes de sensores. Um resultado esperado nessa direção seria a definição de uma linguagem de programação com primitivas mais próximas das necessidades das aplicações, ou a construção de primitivas específicas para um determinado domínio.

Nossa proposta inicial para explorar as possibilidades de combinar comunicação assíncrona e gerência cooperativa de tarefas nos dispositivos que formam as redes de sensores foi integrar a linguagem Lua com a biblioteca AVR Libc (AVRLibc) (um sub-conjunto do padrão ANSI-C definido para os microcontroladores AVR e usado no núcleo do sistema operacional TinyOS). Com base nessa integração, poderíamos usar as mesmas facilidades de Lua que usamos para implementar o ambiente LuaRPC (em particular a construção de

co-rotinas). Entretanto, devido às restrições de espaço de memória das plataformas mais comumente usados com o TinyOS, decidimos adotar uma abordagem mais simples, e com custo computacional menor, implementando diretamente a construção de co-rotinas dentro do TinyOS. Uma direção de pesquisa é retomar a proposta inicial considerando outras plataformas usadas para redes de sensores, nas quais as restrições de espaço de memória são menos severas. Outra direção de pesquisa é explorar as características de simplicidade, portabilidade e tipagem dinâmica da linguagem Lua para permitir carga e adaptação dinâmica de aplicações no contexto de redes de sensores.

Os resultados obtidos embutindo co-rotinas dentro do sistema TinyOS sugere a possibilidade de investigar o uso de co-rotinas como infra-estrutura básica para um modelo de concorrência em sistemas operacionais para redes de sensores. Nessa direção, o passo inicial seria aprofundar o estudo sobre sistemas operacionais embutidos e sobre modelos de concorrência com custo computacional reduzido. A necessidade crucial de otimizar o uso dos recursos computacionais no contexto de redes de sensores, e ao mesmo tempo simplificar o modelo de programação das aplicações para essas redes são exigências básicas que ainda requerem soluções mais aprimoradas.