Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

JSensor: Uma plataforma paralela e distribuída para simulações de redes de sensores

Aluno: Danniel Hugo Ribeiro
Matricula: 08.1.4135

Orientador: Joubert de Castro Lima

Ouro Preto
2 de julho de 2011

Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

JSensor: Uma plataforma paralela e distribuída para simulações de redes de sensores

Relatório de atividades desenvolvidas apresentado ao curso de Bacharelado em Ciência da Computação, Universidade Federal de Ouro Preto, como requisito parcial para a conclusão da disciplina Monografia I (BCC390).

Aluno: Danniel Hugo Ribeiro
Matricula: 08.1.4135

Orientador: Joubert de Castro Lima

Ouro Preto
2 de julho de 2011

Resumo

Apresentamos uma proposta de desenvolvimento de um simulador de algoritmos de redes de sensores que opere de forma paralela e distribuída. Esta terá por finalidade auxiliar na validação de modelos de pesquisas na área de redes de sensores diminuindo o tempo necessário para simular, por exemplo, cenários onde tem-se milhares de nós.

Palavras-chave: Simuladores de Redes. Redes de Sensores. Algoritmos Paralelos. Algoritmos Distribuídos.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

O mundo ao nosso redor possui uma variedade de fenômenos que podem ser descritos por algumas grandezas, como temperatura, pressão e umidade, que podem ser monitorados por dispositivos com poder de sensoriamento, processamento e comunicação. O conjunto desses dispositivos é conhecido na literatura como Rede de Sensores [8]. Pesquisas nesta área utilizam de simulações síncronas e assíncronas para avaliar o impacto de novos modelos a serem propostos de distribuição, roteamento, comunicação ou redução de dados, nos mais variados tipos de redes. Desta forma, Simuladores como o Network Simulator [4] ou o Sinalgo [6] são geralmente utilizados para esses fins.

No entanto, algumas aplicações podem utilizar milhares de nós e tais simuladores podem não suportar tamanha quantidade, o que inviabiliza a validação de algumas soluções propostas e, consequentemente, sua concepção em ambientes reais.

Nesta direção, faz-se necessário a utilização de simuladores que sejam capazes de executar com o menor tempo possível, de forma paralela e distribuída [10][5], simulações em larga escala.

2 Justificativa

A execução deste trabalho fomentará o interesse por questões gerais na área de redes de sensores, sistemas embarcados e automação e algoritmos paralelos. Além da área de redes de sensores, esse projeto abrange as áreas de sistemas distribuídos, computação móvel e todas as áreas relacionadas com redes de computadores. Para o desenvolvimento do mesmo podemos considerar algumas áreas de atuação diretamente correlacionadas:

- *Algoritmos paralelos:* Esta subárea engloba a etapa de soluções para simulação de redes de sensores em paralelo, especificamente o projeto e análise dos algoritmos de simulação existentes no mercado.
- *Algoritmos distribuídos:* Essa subárea engloba a etapa de soluções para simulação de forma distribuída.
- *Otimização:* Essa subárea engloba a etapa de soluções para executar simulações em tempo hábil utilizando ao máximo os recursos disponíveis.

3 Objetivos

3.1 Objetivo geral

- Desenvolver um simulador para redes de sensores que possa executar simulações de forma paralela e distribuída.

3.2 Objetivos específicos

- Elaborar modelos de simulação síncrona e assíncrona que possam ser executados em paralelo e de forma distribuída.
- Desenvolver estratégias que nos permitam executar simulações do tipo síncronas e assíncronas.
- Criar uma interface amigável que permita a visualização dos experimentos em tempo real.
- Executar simulações em larga escala para validar a eficiência do simulador e comparar o resultado obtido com a utilização do JSensor com os outros simuladores do mercado.

4 Metodologia

A metodologia adotada no desenvolvimento do JSensor será:

- Estudar implementações já existentes do funcionamento de uma rede de sensores.
- Criar diagramas de classe para a construção do sistema.
- Implementar o sistema utilizando a linguagem de programação Java [7] juntamente com a linguagem de marcação XML [2]
- Realização de testes para validação do simulador.

4.1 Tecnologias Utilizadas

O sistema será desenvolvido utilizando as tecnologias Java [7] para montar o núcleo do sistema. Além disso, será utilizada também a linguagem de marcação XML [2] para auxiliar na manipulação de configurações a serem definidas pelo usuário.

4.2 Organização do Sistema

O sistema será dividido em camadas como descrito a seguir:

- *Núcleo:* O núcleo do sistema é o componente central do simulador. Nele serão implementadas as funções que permitirão que as simulações sejam executadas de forma paralela e distribuída. Esse módulo é totalmente transparente ao usuário final.
- *Interface Gráfica:* A interface gráfica, ou GUI [9], é o componente onde o usuário do sistema pode interagir com os elementos da rede, bem como controlar o fluxo da simulação (parar, criar nós, controlar o tempo de atualização da GUI, etc.).
- *Projetos:* Este módulo permite que o usuário crie seu próprio modelo de simulação, especificando por exemplo o tipo de nó, "timers", forma de comunicação e detecção, etc.

4.3 Testes

Após o desenvolvimento terá início à etapa de testes. Nesta, o sistema será submetido a vários tipos de simulação, procurando explorar tanto a parte síncrona quando assíncrona do simulador. Além disso, serão feitos testes variando o número de processadores e de máquinas envolvidos para testar a eficiência do mesmo comparada à de outros simuladores do mercado.

5 Desenvolvimento

Nesta seção, será mostrado como está o andamento do desenvolvimento do projeto bem como as etapas concluídas.

5.1 Etapas concluídas

5.1.1 Análise de trabalhos correlatos

Foi feito um estudo dos simuladores existentes no mercado para se obter uma base para a contrução do simulador. Os esforços foram voltados para dois *softwares* específicos: O Network Simulator 3 (NS-3) [4] e o Sinalgo [6].

1. *Network Simulator 3 (NS-3)*
   É um simulador assíncrono para sistemas de internet. Ele é escrito em C++/Python e as aplicações nele são desenvolvidas nas mesmas. É gratuito e de código livre e está sobre a licença GNU GPLv2 [3]. Pode ser utilizado tanto no Windows quanto em Sistemas Operacionais baseados em Linux.
   O NS-3 foca seus esforços no realismo, buscando deixar as simulações o mais próximo possível de modelos reais. Assim sendo, ele possui implementados internamente, vários protocolos e camadas das redes habituais, como por exemplo o TCP-IP.
   Além de não possuir interface gráfica integrada, a arquitetura do NS-3 não está preparada para arquiteturas multi-processadas ou distribuída, sendo assim, o tempo de simulação e a quantidade de nós que podem ser utilizados em uma simulação depende diretamente da quantidade de memória física disponível.

2. *Sinalgo*
   O Sinalgo é um *framework* para teste e validação de algoritmos de redes. Ao contrário de outros simuladores disponíveis, o Sinalgo foca na verificação de novos algoritmos. Sendo assim, as simulações não implementa protocolos específicos das camadas de rede. Desenvolvido em Java, ele é também um software livre e de código aberto, sob a licença BSD [1]. Suporta simulações do tipo síncrona e assíncrona e tem suporte à visualização em 2D e 3D.
   Assim como o NS-3, o Sinalgo também não tem suporte a arquiteturas multi-processadas e distribuída, mas seus fabricantes garantem que simulações com até cem mil nós podem ser executadas em tempo hábil.

5.1.2 Escolha de um software base

Após uma análise de ambos, decidiu-se que a implementação o JSensor seria baseada no Sinalgo, pelos seguintes motivos:

1. O Sinalgo é feito em Java.
2. É mais fácil de se desenvolver novas aplicações nele do que no NS-3.
3. Possui interface, o que facilita na visualização da rede a ser simulada.
4. Não se prende a implementações das camadas e subcamadas de rede.
5. Possui suporte a simulações síncronas e assíncronas.

5.1.3 Modelagem e implementação do núcleo do JSensor

O fluxo de execução do JSensor é mostrado na Figura 1.

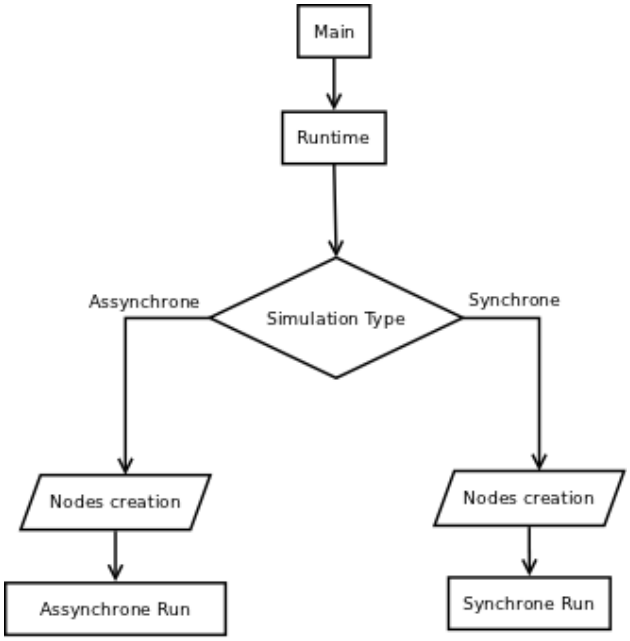

Figura 1: Fluxo de execução do JSensor

O programa começa sendo executado do Main e um *runtime* é criado. Ele avalia qual tipo de simulação será utilizada (síncrona ou assíncrona, dependendo da escolha do usuário). A partir deste ponto o usuário pode criar os nós sensores, tantos quanto ele julgar necessário e controlar o fluxo de execução da simulação através da GUI.

5.2 Desenvolvimento da GUI

O desenvolvimento da GUI se encontra em andamento. No entanto, algumas funções para visualização e criação de nós sensores já foram desenvolvidas.

5.2.1 Interface de criação de nós

Esta interface é responsável pela criação dos nós. Nela o usuário pode especificar a quantidade de nós a serem gerados. Pode-se também especificar alguns parâmetros auxiliares como por exemplo a forma como eles serão dispostos na rede, o tipo de comunicação, o tipo de mobilidade e conectividade. Todos estes parâmetros podem ser implementados e utilizados pelo próprio usuário. A interface de criação de nós é mostrada na Figura 2.

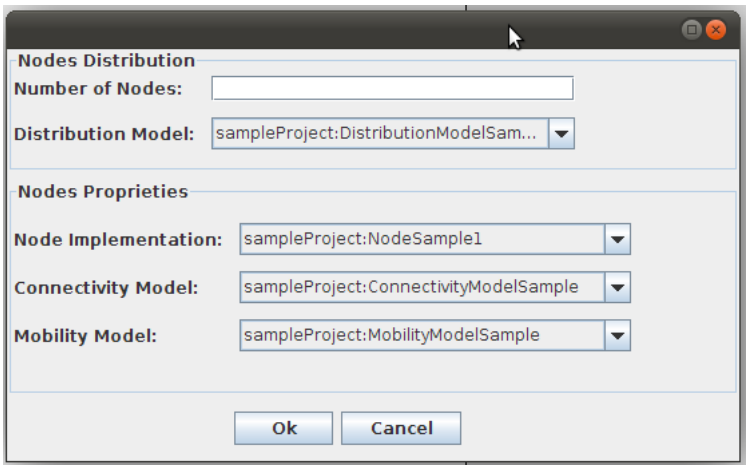

Figura 2: Interface de criação de nós

5.2.2 Interface de controle e visualização de simulações

Responsável por controlar e mostrar o andamento das simulações. O usuário pode através dela iniciar e pausar uma determinada tarefa através do botão `Start`. Pode-se também especificar o número de *rounds* e de atualização da GUI a serem executados na simulação. A Figura 3 mostra um exemplo desta interface. O painel central representa os nós (círculos em preto) e suas respectivas disposições no cenário. À direta tem-se o painel de controle onde o usuário controla o andamento da simulação. No topo da interface encontra-se o menu de criação e destruição de nós.

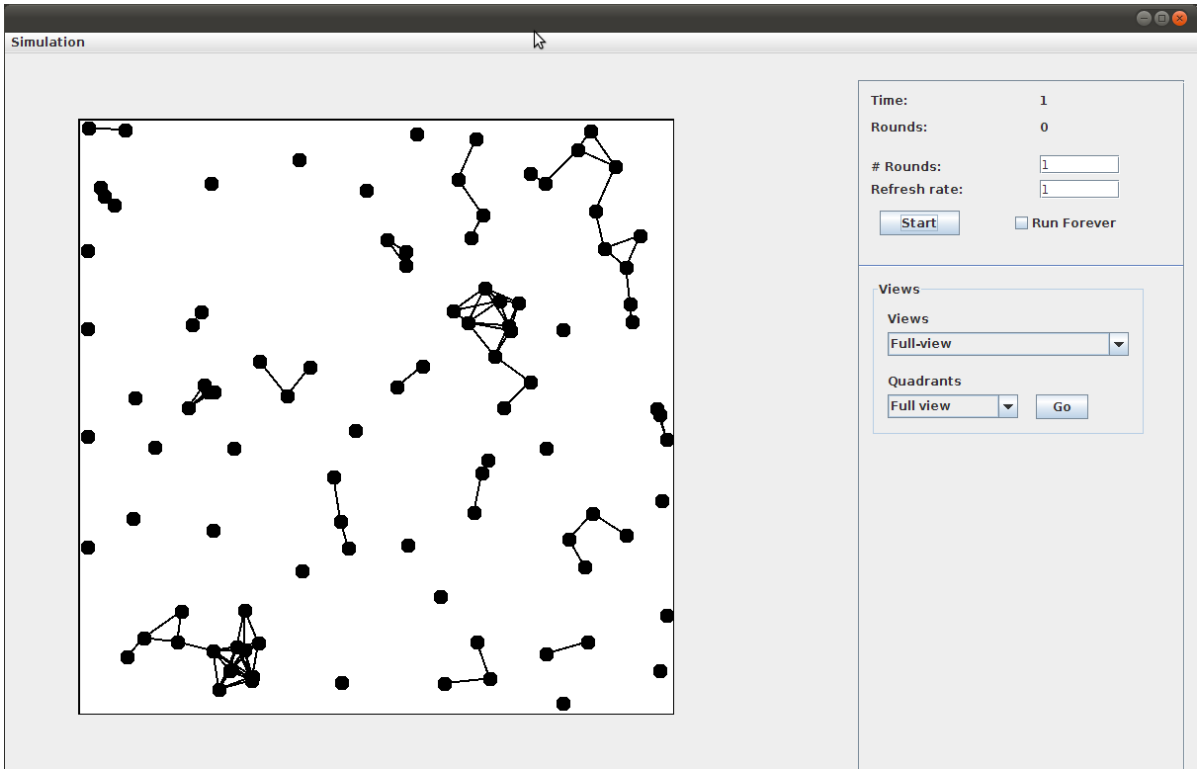

Figura 3: Interface visualização e controle de simulações

5.3 Desenvolvimento da versão multiprocessada

O desenvolvimento da versão multiprocessada, ou paralela, do JSensor é também uma etapa em andamento. Esta é dividida em duas subetapas: Simulação Síncrona Multiprocessada e Simulação Assíncrona Multiprocessada. Em sua versão atual, o JSensor possui implementações em ambas as subetapas.

5.3.1 Simulação Síncrona Multiprocessada

Uma simulação do tipo síncrona é um tipo de simulação onde o fluxo de execução é baseado em *rounds*. A cada *round* é feita uma iteração sobre todos os nós da rede executando ações pré-definidas. O Algoritmo 1 representa as etapas da execução de uma simulação síncrona. A cada *round* o algoritmo incrementa o tempo global (linha 2). Depois, para cada nó pertencente à rede ele executa três etapas:

- Move o nó, de acordo com a implementação do usuário de um modelo de mobilidade (linha 4). Este modelo pode ser simples, como por exemplo encontrar uma posição aleatória no terreno ou até mesmo algo mais complexo baseados na velocidade, tempo e distância.

- Atualiza as conexões do nó (linha 5). Nesta etapa o nó descobre quem são seus vizinhos e atualiza suas conexões, respeitando as regras expressas no modelo de conectividade.

- Executa o método Step do nó (linha 6). Este método executa uma sequência de ações pré-definidas que podem ser implementadas pelo usuário. São elas:
  - *Pré-Step*: Primeira ação a acontecer quando o método Step é invocado.
  - *Neighborhood Change*: Acontece quando a vizinhança do nó sofreu alterações.
  - *Handle Timers*: Executa os eventos de Tempo associados ao nó.
  - *Handle Messages*: Executa ações quando há mensagens na caixa de entrada do nó.
  - *Post Step*: Ação que ocorre ao final do método Step.

Algoritmo 1 Algoritmo de Simulação Síncrona

Entrada: *_nodes* – Conjunto de todos os nós da rede, *_rounds* número de rounds a serem executados
1: **para** $j \leftarrow 0$ até *_rounds* **faça**
2: Incremente o tempo global em 1
3: **para todo** *node* em *_nodes* **faça**
4: Mova *node*. "De acordo com seu respectivo modelo de mobilidade"
5: Atualize conexões de *node*. "De acordo com seu modelo de conectividade"
6: Execute o método Step de *node*.
7: **fim para**
8: **fim para**

No JSensor a simulação síncrona é executada de forma paralela para utilizar de forma mais eficiente os recursos da CPU. O Algoritmo 2 demonstra o Algoritmo 1 sendo executado de forma concorrente. Na linha 4 do Algoritmo 2 acontece a criação de um conjunto de *threads* de tamanho especificado pelo usuário. É associado a cada nova *thread*, um subconjunto de nós da rede (linha 5). Cada *thread* executa então, os mesmos passos do Algoritmo 1, porém de forma paralela (linhas 7 a 15).

Algoritmo 2 Algoritmo do fluxo de execução síncrono paralelo

Entrada: *_nodes* – Conjunto de todos os nós da rede, *_n* quantidade de nós ,
_cores – Quantidade de núcleos ou de Threads a serem criadas, *_rounds* número de rounds a serem executados
1: **para** $j \leftarrow 0$ até *_rounds* **faça**
2: Incremente o tempo global em 1
3: **para** $i \leftarrow 0$ até *_cores* **faça**
4: Crie Thread $T[i]$
5: Delege à Thread $T[i]$ o subconjunto $S[i] \in$ *_nodes* de tamanho (*_n*/*_cores*)
6: **fim para**
7: **para** $i \leftarrow 0$ até *_cores* **faça**
8: Movimente os nós de $S[i]$ na thread $T[i]$
9: **fim para**
10: **para** $i \leftarrow 0$ até *_cores* **faça**
11: Atualize conexões dos nós de $S[i]$ na thread $T[i]$
12: **fim para**
13: **para** $i \leftarrow 0$ até *_cores* **faça**
14: Execute o método Step dos nós de $S[i]$ na thread $T[i]$
15: **fim para**
16: **fim para**

5.3.2 Simulação Assíncrona Multiprocessada

Em uma simulação do tipo assíncrona um nó reage sob três tipos de eventos: Tempo, Detecção e Envio.
Eventos do tipo Tempo são aqueles programados para ocorrer em um determinado

tempo t. Eventos do tipo Detecção acontecem quando um nó está perto de um fenômeno e o detecta (fogo, enchente, desmoronamento). Já o evento do tipo Envio, acontece quando um nó envia uma mensagem a outro(s) nó(s) da rede.
O Algoritmo 3 mostra o fluxo de execução do modo assíncrono de forma paralelo. De forma semelhante ao Algoritmo 2, cria-se um conjunto de *threads* e a elas são associados subconjuntos de nós (linhas 2 a 5). Cada *thread* avalia a caixa de entrada de seus respectivos nós e manipula as mensagens recebidas através do método *handleMessages()* se necessário (linhas 6 a 11). Por fim o método *step* é invocado, tendo como parâmetro *ti* que é o tempo corrente (linhas 12 a 14).

Algoritmo 3 Algoritmo do fluxo de execução assíncrono paralelo

Entrada: *_nodes* – Conjunto de todos os nós da rede,*_n* quantidade de nós
,*_cores* – Quantidade de núcleos ou de Threads a serem criadas,*_seconds* número de segundos a serem executados

```
1: para ti ← 0 até _seconds faça
2:     para i ← 0 até _cores faça
3:         Crie Thread T[i]
4:         Delege à Thread T[i] o subconjunto S[i] ∈ _nodes de tamanho (_n/_cores)
5:     fim para
6:     para i ← 0 até _cores faça
7:         Avalie caixa de entrada dos nós de S[i] na thread T[i]
8:         se Caixa de entrada não está vazia então
9:             Manipule as mensagens recebidas do nó
10:        fim se
11:    fim para
12:    para i ← 0 até _cores faça
13:        Execute o método Step(ti) dos nós de S[i] na thread T[i]
14:    fim para
15:    Incremente o tempo global em 1
16: fim para
```

O Algoritmo 4 representa como os eventos são tratados. Na linha 1 é retirado um evento da pilha de eventos do nó. Somente são retirados eventos cujo tempo de ocorrência é menor ou igual ao tempo máximo. Descobre-se o tipo do evento, e então ele é executado (linhas 2 a 8)

Algoritmo 4 Algoritmo do método Step na execução assíncrona

Entrada: *_node* – Nó avaliado, *_ti* – Tempo de ocorrência máximo dos eventos

```
1: Retire um evento E de _node tal que o tempo de ocorrência T(E) é menor ou igual à _ti
2: se E é um evento tipo Tempo então
3:     Execute E como tipo Tempo
4: senão se E é um evento tipo Detecção então
5:     Execute E como tipo Detecção
6: senão
7:     Execute E como tipo Envio
8: fim se
```

6 Trabalhos Futuros

Como trabalhos futuros relacionados a este projeto tem-se:

- Aperfeiçoamento da interface.
- Escrita de um artigo sobre a versão paralela do JSensor.
- Implementação do módulo distribuído do JSensor.

- Escrita de um novo artigo sobre a versão distribuída do JSensor.
- Realização de testes.
- Elaboração da monografia.
- Apresentação da monografia para a banca examinadora.

7 Cronograma de atividades

As atividades a serem desenvolvidas são:

1. Estudo dos simuladores presentes no mercado.
2. Estudo de algoritmos paralelos.
3. Implementação da GUI.
4. Implementação do módulo síncrono paralelo do JSensor.
5. Implementação do módulo assíncrono paralelo do JSensor.
6. Implementação do módulo distribuído do JSensor.
7. Realização de testes.
8. Elaboração do relatório de atividades.
9. Apresentação do relatório de atividades.
10. Elaboração da monografia.
11. Apresentação da monografia para a banca examinadora.
12. Escrita de artigos científicos.

Na Tabela 1, apresenta-se o cronograma de atividades.

Atividades	**Jan**	**Fev**	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**	**Ago**	**Set**	**Out**	**Nov**
1	X	X									
2	X	X									
3	X	X	X	X	X	X	X				
4			X	X	X						
5					X	X	X				
6							X	X	X		
7			X	X	X	X	X	X	X		
8				X	X	X					
9						X					
10								X	X	X	X
11											X
12			X	X	X	X	X	X	X	X	X

Tabela 1: Cronograma de Atividades.

Referências

[1] BSD license. http://www.opensource.org/licenses/bsd-license.php, 2011. Visitado em 2 de julho de 2011.

[2] Extensible markup language (XML). http://www.w3.org/XML/, 2011. Visitado em 2 de julho de 2011.

[3] GPL - 2.0. http://www.gnu.org/licenses/gpl-2.0.html, 2011. Visitado em 2 de julho de 2011.

[4] The ns-3 network simulator. http://www.nsnam.org/index.html, 2011. Visitado em 2 de julho de 2011.

[5] Producer-consumer problem. http://en.wikipedia.org/wiki/Producer-consumer_problem, 2011. Visitado em 2 de julho de 2011.

[6] Sinalgo - simulator for network algorithms. http://www.disco.ethz.ch/projects/sinalgo/, 2011. Visitado em 2 de julho de 2011.

[7] What is java? http://www.java.com/en/download/whatis_java.jsp, 2011. Visitado em 2 de julho de 2011.

[8] Ian F. Akyildiz, WellJan Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[9] Wilbert O. Galitz. *The Essential Guide to User Interface Design*. Wiley, 2 edition, 2002.

[10] Andrew S. Tanenbaum and Maarte Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2 edition, 2007.