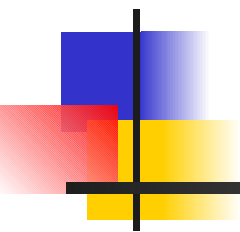


Um Arcabouço para Suporte a Reconfiguração Dinâmica em Ambiente Java



Ricardo Koji Ushizaki
Orientador: Prof. Dr Fabio Kon

Abril/2005



Agenda

- Três partes:
 - Reconfiguração Dinâmica
 - Trabalhos Relacionados
 - Arcabouço Java



Primeira Parte

- **Reconfiguração Dinâmica**
- Trabalhos Relacionados
- Arcabouço Java

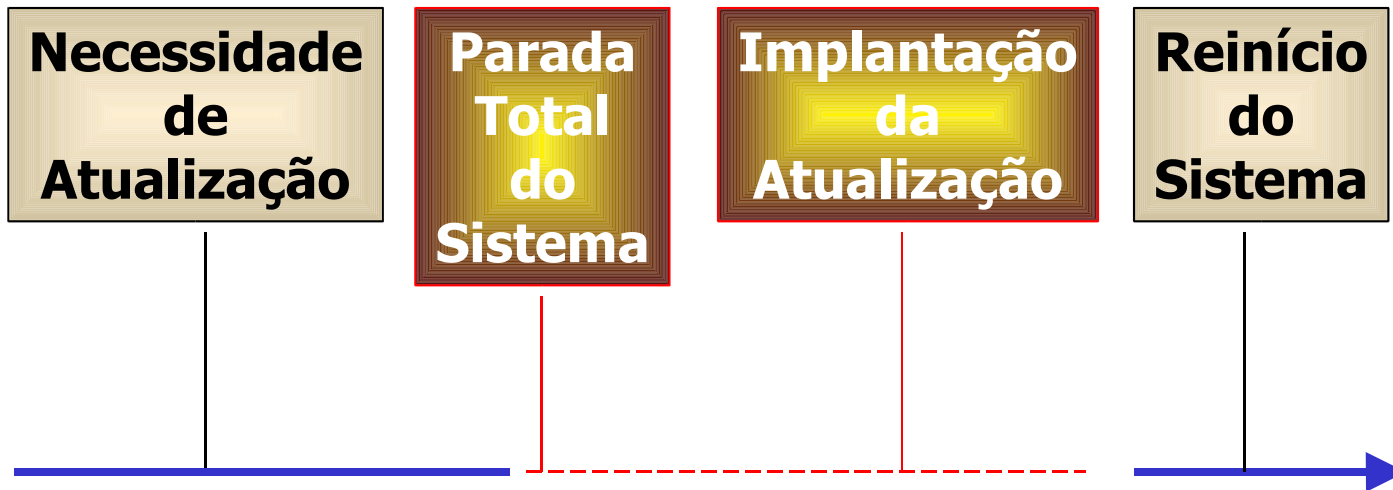


Reconfiguração Dinâmica

- Sistemas computacionais em constante evolução
- Programação orientada a objetos
- Baseada em componentes
- Evolução requer atualizações do sistema



Cenário Típico

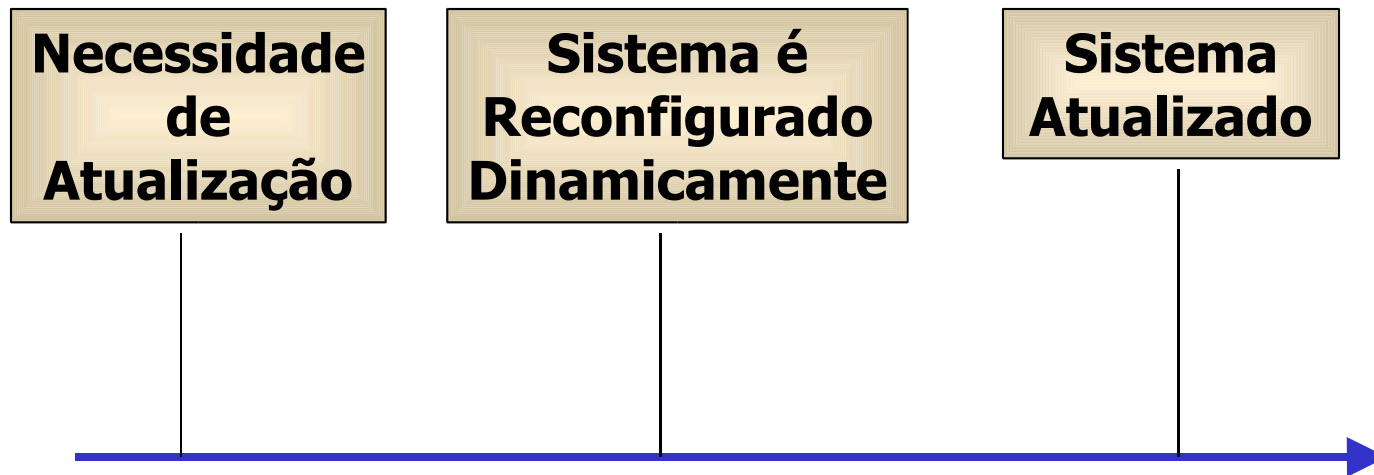




Problemas

- Paradas do Sistema significam:
 - Prejuízos financeiros para empresas;
 - Vidas em perigo em sistemas críticos (hospitais).
- É necessário atualizar dinamicamente os sistemas

Cenário com Reconfiguração Dinâmica





Histórico

- Primeiros trabalhos de reconfiguração dinâmica surgiram na década de 80
- Formalizaram técnicas e identificaram obstáculos para reconfiguração dinâmica



Objetivos da Reconfiguração Dinâmica

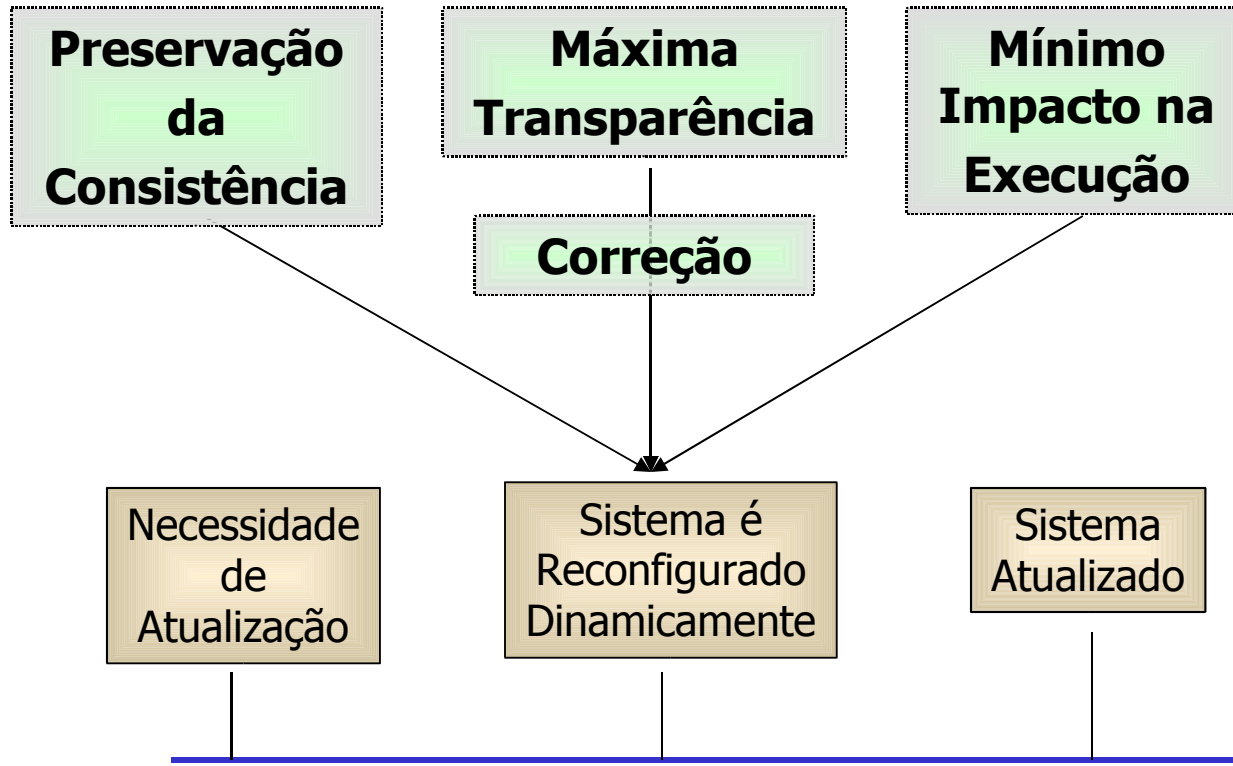
- Permitir a evolução do sistema em tempo de execução
- Pouco ou nenhum impacto negativo no seu desempenho



Operações da Reconfiguração Dinâmica

- Adição e Remoção
 - Adicionar novo componente
 - Remover componente
- Substituição
 - Substituir implementação
- Migração
 - Mover componente

Requisitos



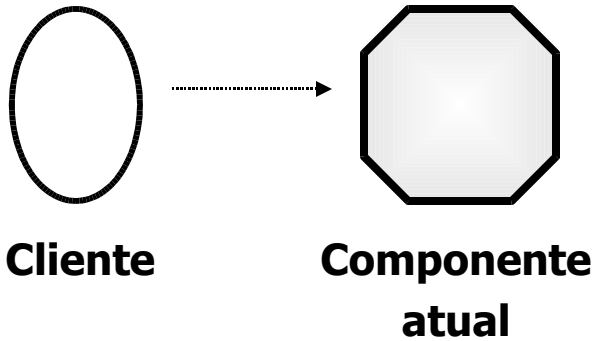


Aspectos Importantes

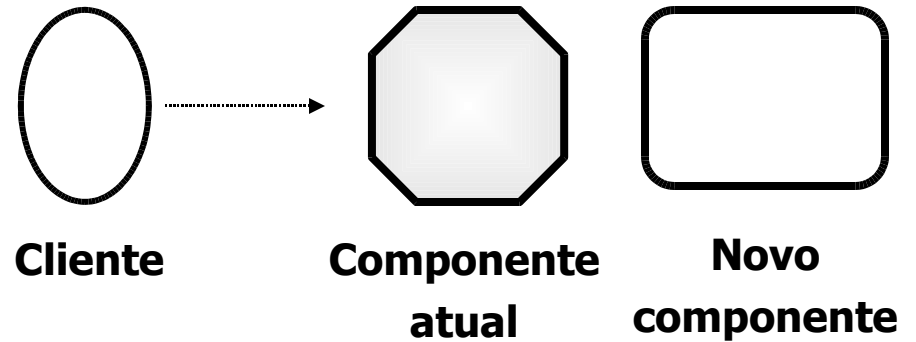
- Aplicações possuem estado
 - Deve ser mantido correto e consistente
- Garantir a Transferência de Estado
 - Estado deve ser migrado da implementação atual para a nova a ser instalada
 - Transferência de Estado não deve corromper o sistema

Exemplo

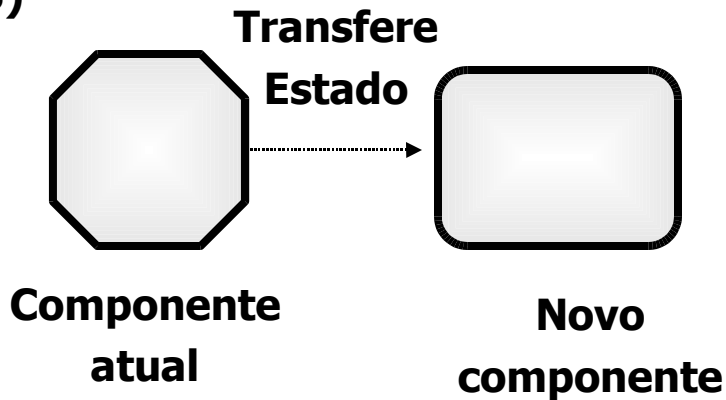
1)



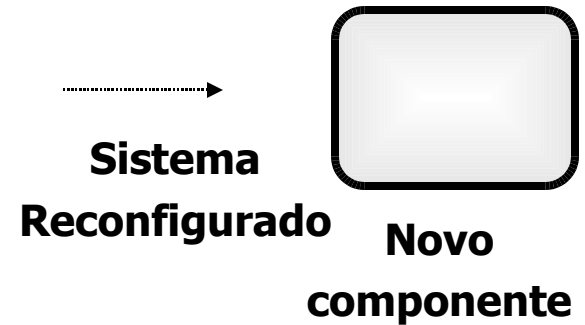
2)



3)



4)

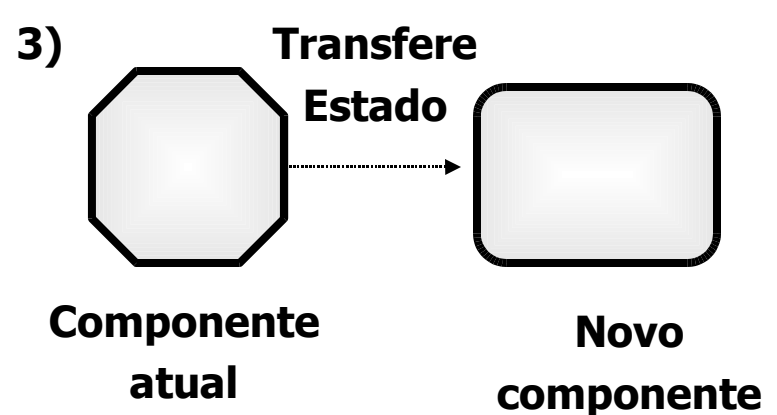
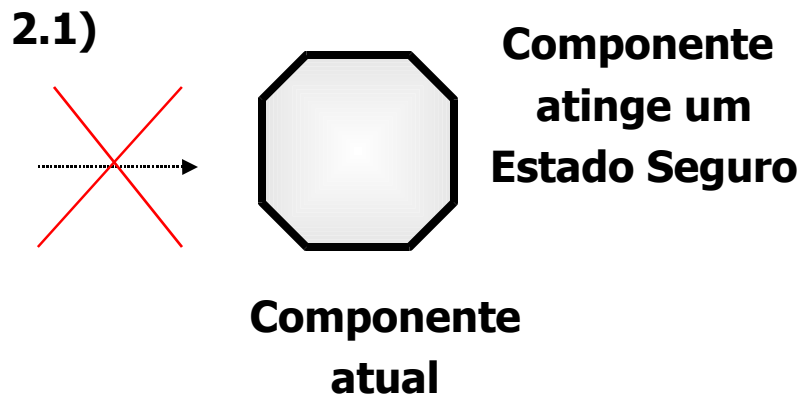
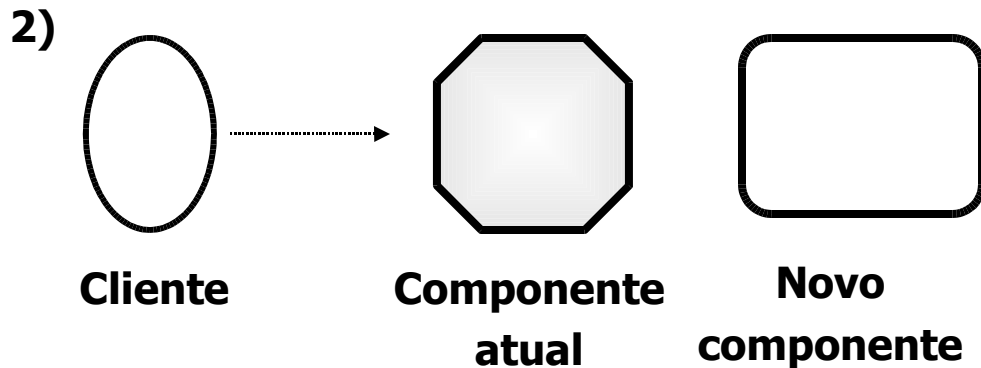




Estado Seguro

- É necessário preservar a consistência do sistema após a reconfiguração
- Para ocorrer a transferência de estado:
 - Componentes devem estar em um **estado seguro**
 - Garantir que permaneçam nesse estado durante a reconfiguração

Atingindo Estado Seguro





Preservar Consistência

- Sistema deve estar em um estado correto após a reconfiguração
- Estado correto:
 - Integridade estrutural preservada;
 - Partes afetadas continuam interagindo com sucesso;
 - Invariantes do estado preservadas.



Mecanismos

- Foram propostos diversos mecanismos para a reconfiguração dinâmica:
 - Mecanismo de Bloqueio de Chamadas
 - Mecanismo de Indireção de Chamadas
 - Mecanismo de Pontos de Reconfiguração

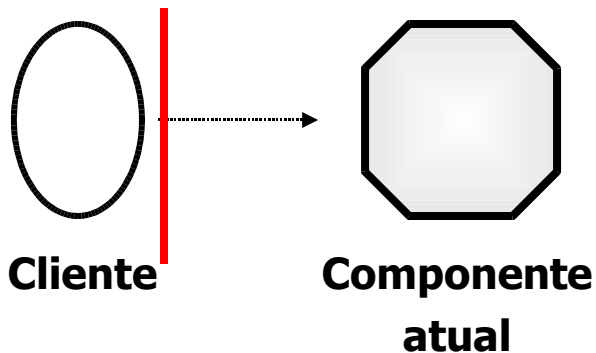


Bloqueio de Chamadas

- Para o componente chegar a um estado seguro:
 - bloquear chamadas a métodos que alterem seu estado
 - Ou abortar chamadas a esses métodos (por exemplo, via Java Exceptions)

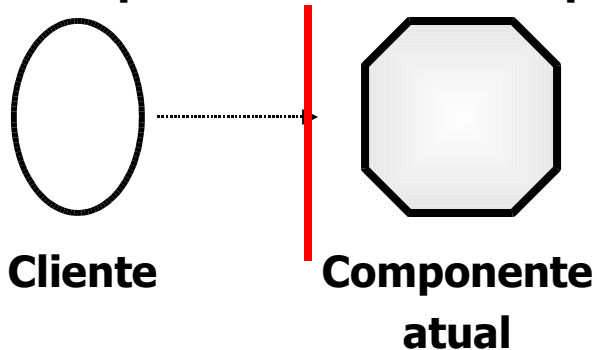
Exemplos de bloqueios

1- Bloquear no lado cliente



- Mais simples de implementar
- Muito intrusiva
- Nem sempre podemos alterar cliente

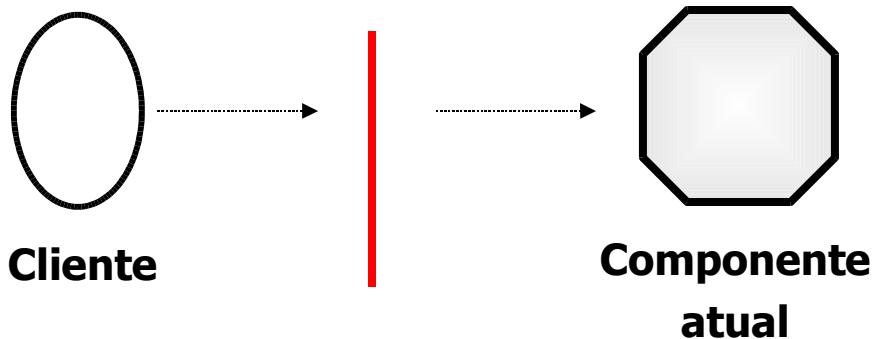
2- Bloquear no lado do componente



- Transparente ao cliente
- Manipular variáveis locais

Exemplos de bloqueios

3- Bloquear entre o cliente e o componente



- Indireção De Chamadas
- Transparente ao cliente
- Toda chamada tem um "passo" a mais

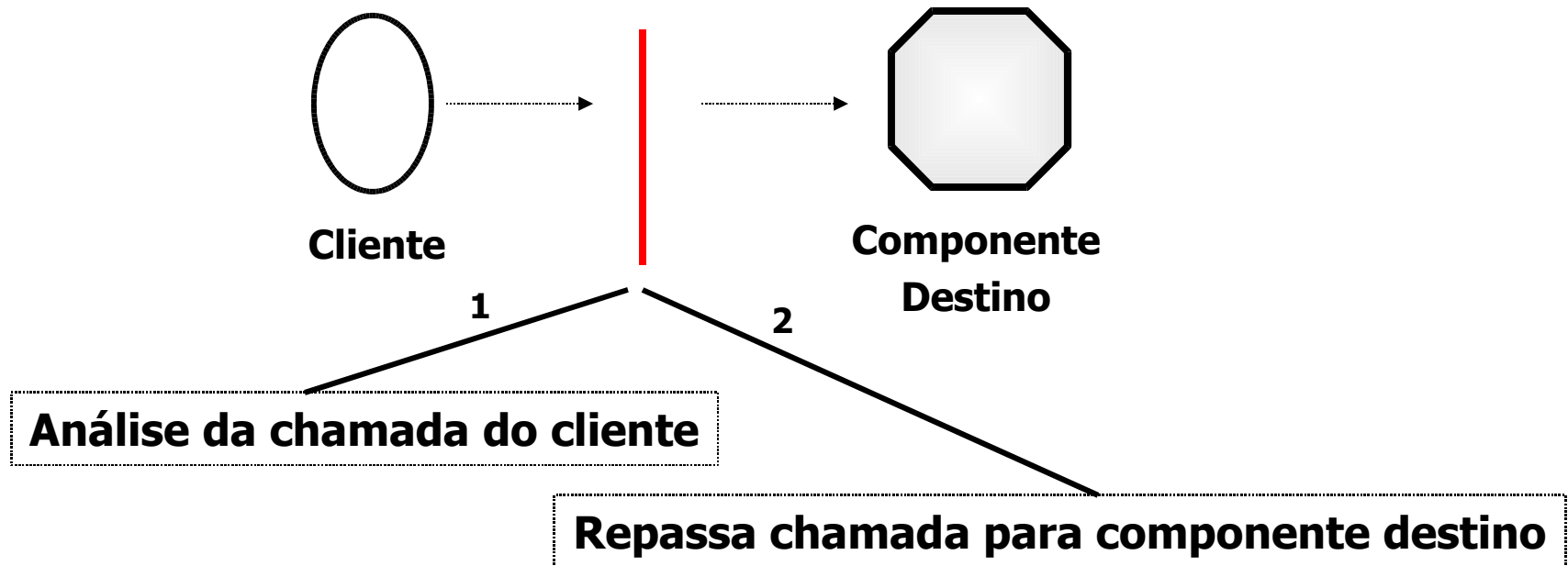


Indireção de Chamadas

- Utiliza mecanismos do middleware
 - CORBA, EJB, .NET
- Redireciona chamadas do cliente para componente
- Existe um serviço intermediário que intercepta toda chamada

Exemplo de indireção

Serviço intermediário intercepta chamada



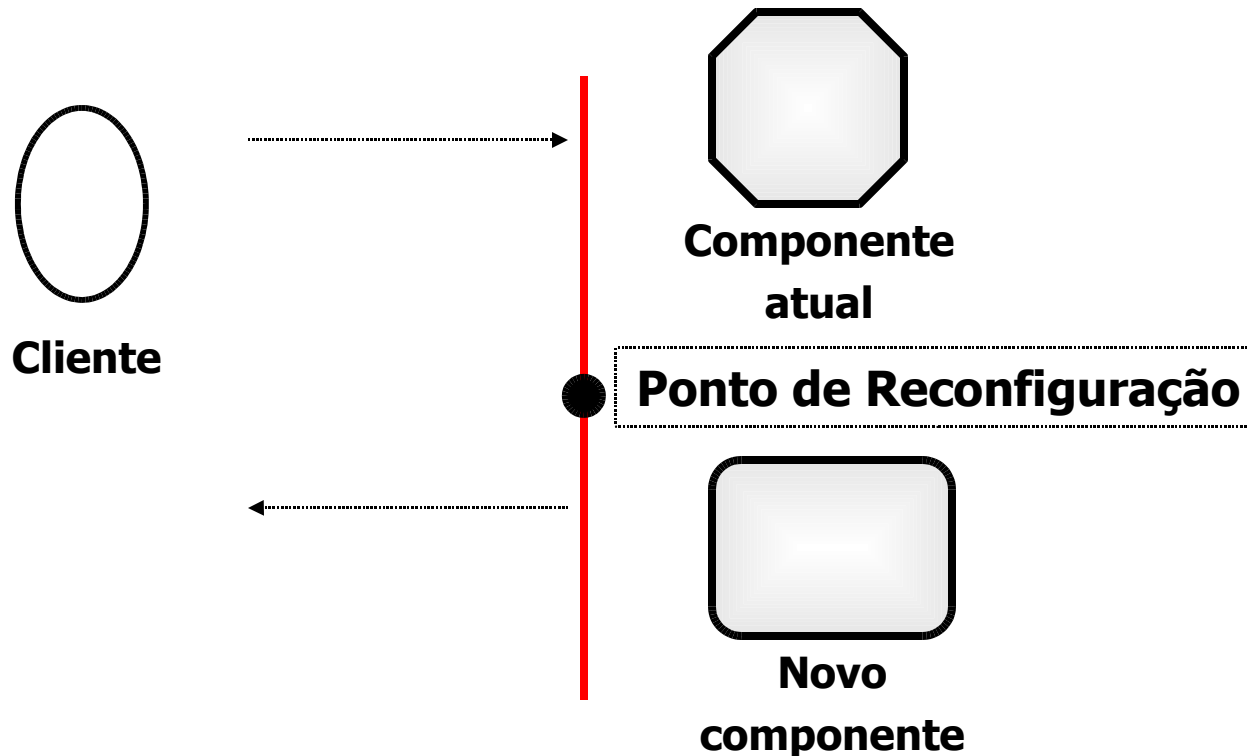


Pontos de Reconfiguração

- Pontos no código seguros para a reconfiguração
- Não precisa esperar que a chamada termine para iniciar a reconfiguração
 - Execução do método é interrompida no ponto de reconfiguração
 - Cliente inicia chamada em um componente e finaliza no novo componente

Exemplo de Ponto de Reconfiguração

Bloquear no lado do componente





Pontos de Reconfiguração

- Vantagens:
 - Atinge estado seguro durante invocação do método
 - Métodos de longa execução não atrasam a reconfiguração



Pontos de Reconfiguração

- Desvantagens:
 - Preservar estado do componente e do método invocado
 - Difícil de implementar:
 - nova implementação deve compartilhar o mesmo comportamento após o ponto de reconfiguração
 - Caso contrário, não é possível mapear contexto



Segunda Parte

- Reconfiguração Dinâmica
- **Trabalhos Relacionados**
- Arcabouço Java



Trabalhos Relacionados - Argus

- Substituição Dinâmica no Argus
 - MIT - 1983 - Toby Bloom
 - Formalizou um modelo de substituição dinâmica utilizando Argus
 - Argus é um sistema distribuído composto por módulos chamados de guardiões
 - Definiu mecanismo para substituir manualmente os guardiões



Trabalhos Relacionados - Conic

- Configuração Dinâmica com Conic
 - Imperial College – 1985 – Jeff e Magee
 - Conic:
 - ambiente de programação
 - ferramentas para compilar, configurar, depurar e executar programas
 - Como configurar dinamicamente sistema em Conic
 - Definem comandos: link, unlink, create e um ConfigurationManager para reconfiguração
 - Termo quiescence: componente em modo passivo



Trabalhos Relacionados - POLYLITH

- POLYLITH

- É um sistema distribuído para ambientes heterogêneos
- Universidade de Maryland – 1990
- Purtillo e Hofmeister incluem suporte para reconfiguração dinâmica no POLYLITH
- Estenderam o trabalho de Jeff e Magee
- Utiliza pontos de reconfiguração para capturar e restaurar estado



Trabalhos Relacionados - Bidan

- Serviço de Reconfiguração Dinâmica para CORBA
 - Bidan em 1998 estendeu a semântica do serviço CORBA de ciclo de vida
 - Criou Dynamic Reconfiguration Manager:
 - Interage com componentes
 - Operação `passivateLink`
 - Interface `RO_Object`



Trabalhos Relacionados

- Reconfiguração Dinâmica de Aplicação Java Baseada em Componentes
 - MIT – 2000 – Ziqiang Tang
 - Define modelo utilizando pontos de reconfiguração
 - Define novas palavras-chaves em Java:
 - `component`, `decode`, `encode`, `fulfills`, `reconfigurables`, `reconfigurable`



Trabalhos Relacionados

- Reconfiguração Dinâmica Transparente para CORBA
 - Almeida et al – 2001
 - Estendem o trabalho de Bidan adicionando suporte para:
 - Chamadas re-entrantes
 - Substituições atômicas de múltiplos objetos
 - Maior transparência no ORB do CORBA
 - Define arquitetura de componentes:
 - Reconfiguration Manager, Location Agent e Reconfiguration Agents



Trabalhos Relacionados

- Sistema Operacional 2K
 - Universidade de Illinois – 2000 – Kon et al
 - Criaram sistema operacional distribuído baseado em componentes
 - Define um Serviço de Configuração Automática para:
 - Gerenciar pré-requisitos
 - Gerenciar dependências dinâmicas entre componentes



Trabalhos Relacionados

- Adaptação Dinâmica de Sistemas Distribuídos
 - IME USP – 2003 – Francisco Silva e Silva
 - Criou arquitetura para adaptação dinâmica
 - Módulos de monitoração do sistema
 - Detecção e análise de eventos
 - Verificação de pré-condições e aplicar reconfiguração dinâmica



Trabalhos Relacionados – JBoss e JMX

- Marc Fleury e Reverbel apresentaram em 2003 a arquitetura JMX do JBoss
- JMX – Java Management Extensions
 - Principal arquitetura do JBoss
 - Gerenciamento dinâmico e monitoração
 - Componentes Mbeans (Managed Beans)
 - Nível de indireção entre clientes e Mbeans favorece a reconfiguração do sistema



Terceira Parte

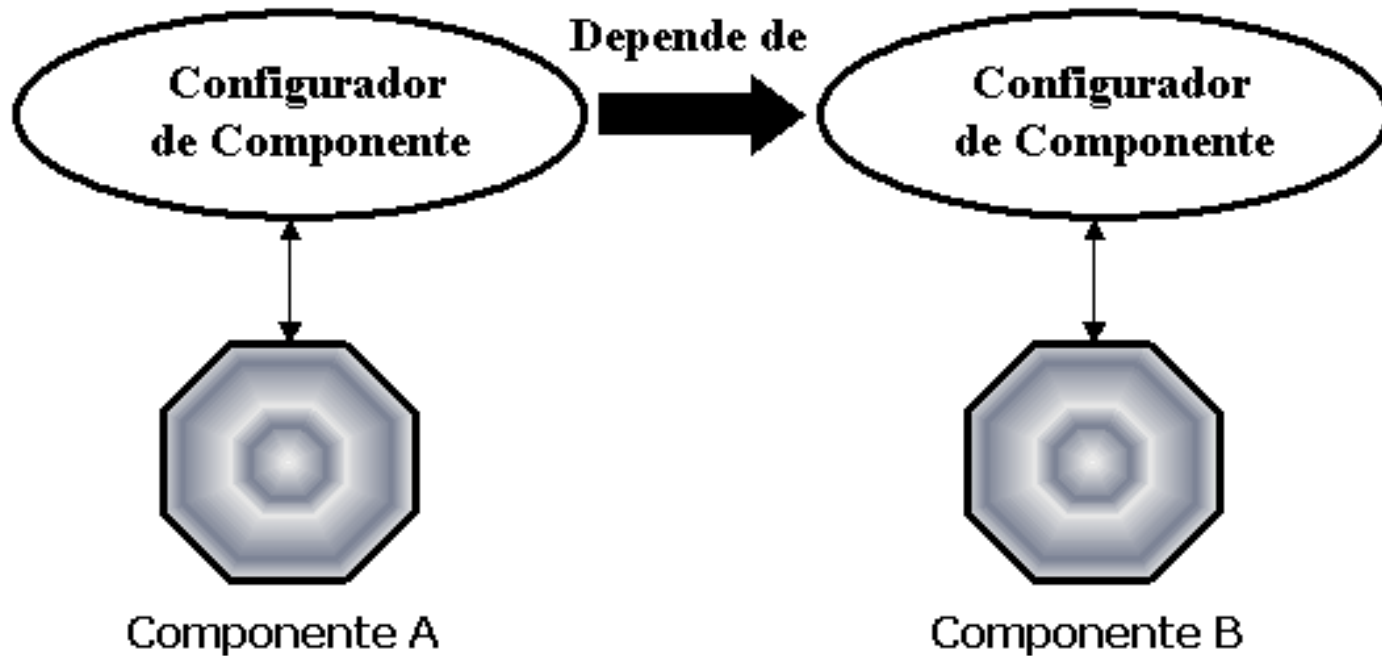
- Reconfiguração Dinâmica
- Trabalhos Relacionados
- **Arcabouço Java**



Configuradores de Componentes

- Modelo seguido pelo nosso arcabouço
 - Objetivo: representar as dependências dinâmicas entre componentes
- Cada componente possui um Configurador de Componente (CC) associado

Modelo





Dependências

- São representadas através de ganchos e clientes
- Quando Componente A depende de B:
 - Anexar CC.B a um dos ganchos do CC.A
 - Adicionar CC.A à lista de clientes do CC.B

Dependências

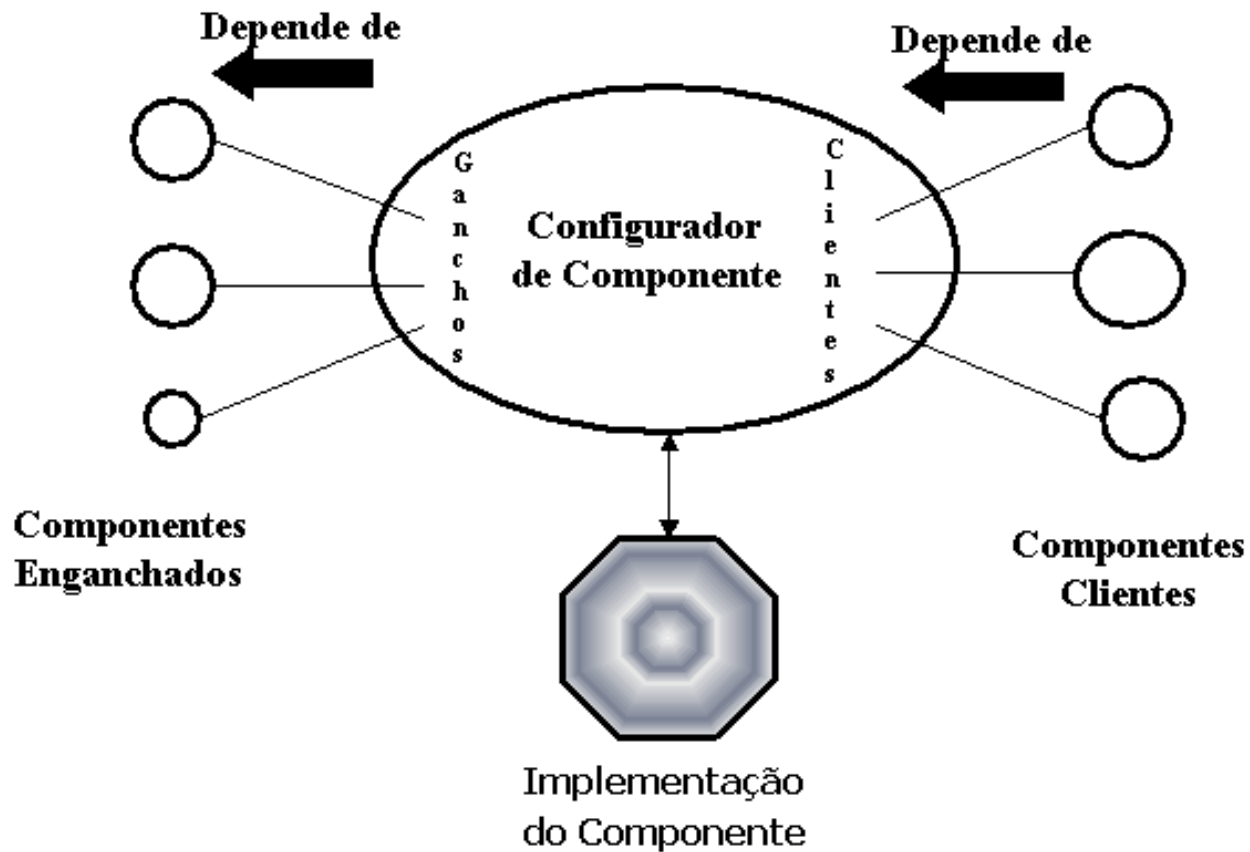
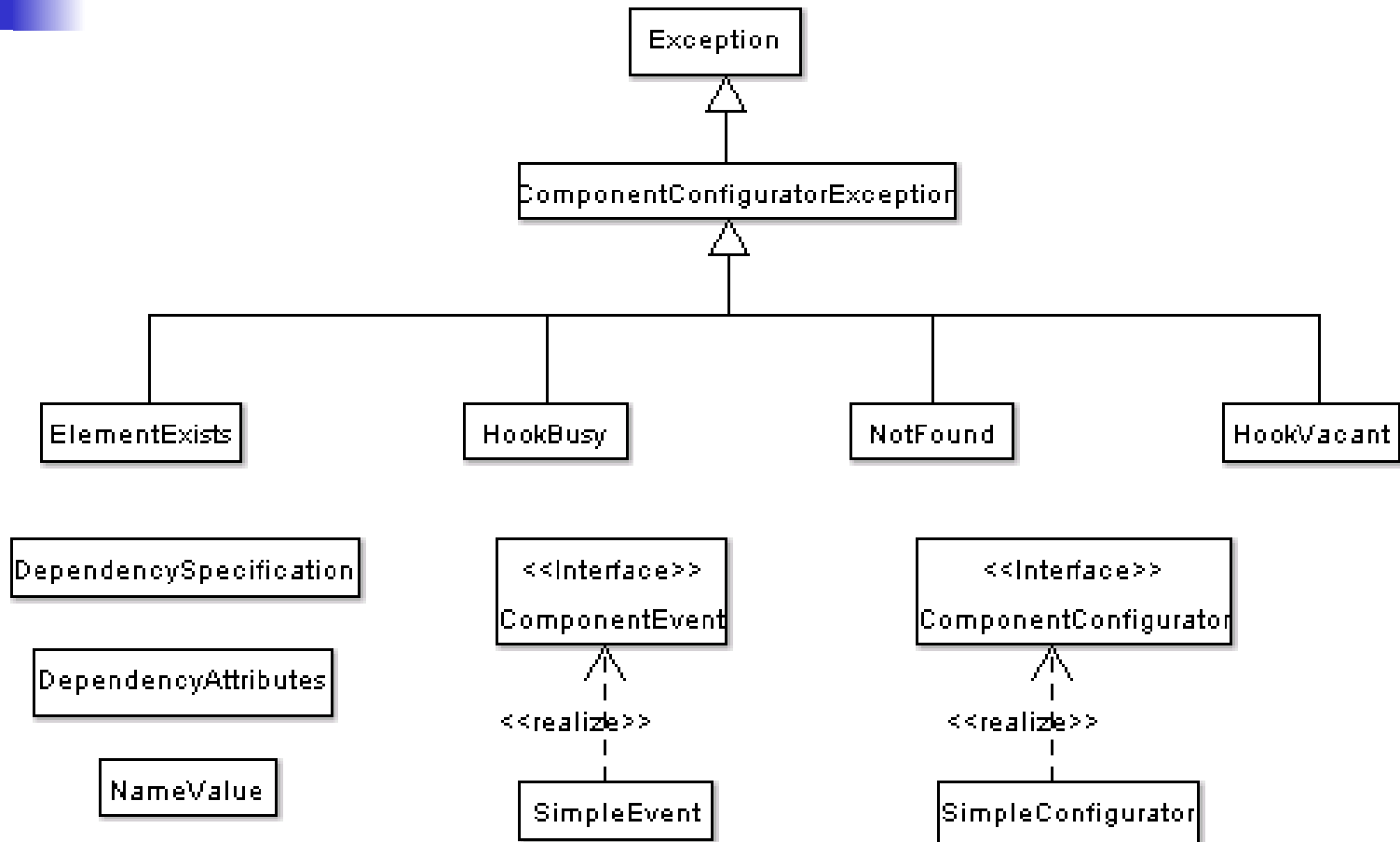


Diagrama de Classes





Extensão do Modelo

- Arcabouço de Objetos Reconfiguráveis
- Extensão dos Configuradores de Componentes
- Interface própria para reconfiguração:
ReconfigurableObject
 - Métodos para exportar e importar o estado
 - Executar a substituição do componente



ReconfigurableObject

- Todo objeto reconfigurável deve implementar essa interface
- Três métodos principais:
 - `prepareReconfiguration()`
 - Exporta o estado
 - `replaceImplementation()`
 - Substituição dinâmica do objeto
 - `initFromROState()`
 - Importa o estado



Transferência de Estado

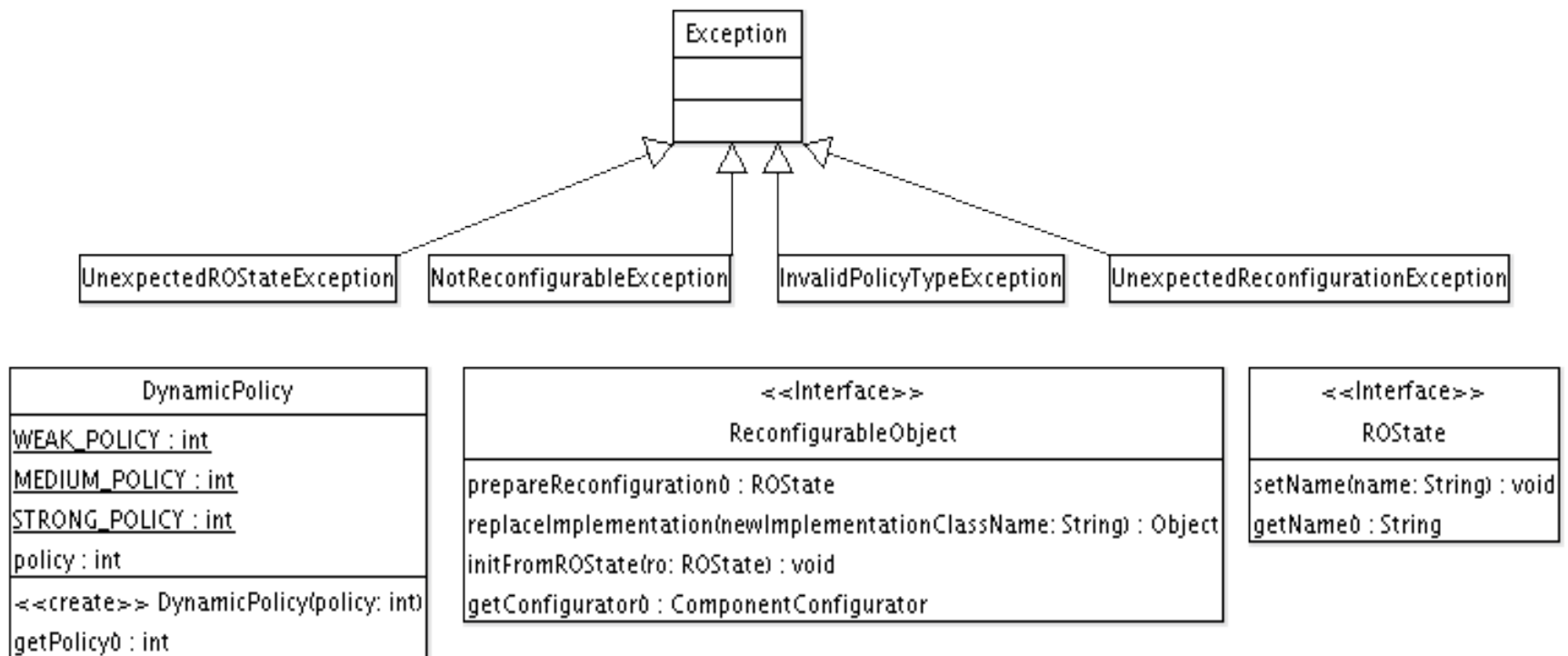
- Exportar e importar o estado através de um `RState`
- Baseado no padrão Memento:
 - Armazena estado interno do componente
 - Estado é recuperado na importação



Substituição de Implementação

- Substitui dinamicamente o componente
- Método `replaceImplementation()`
 - Exporta estado atual
 - Cria nova instância da nova implementação
 - Importa estado na nova instância
 - Atualiza referências

Diagrama de Classes





Políticas de Consistência

- Determinam o comportamento dos componentes durante a reconfiguração
- Indica ao componente quando bloquear e sincronizar chamadas
- Três tipos de políticas:
 - Fraca – não precisa de bloqueios
 - Média – bloqueia parcialmente
 - Forte – sempre bloqueia



Diferenças entre Políticas

- Política de Consistência Fraca:
 - Componentes stateless (sem estado)
- Política de Consistência Média
 - Define partes do componente consistentes
- Política de Consistência Forte
 - Consistência total do estado



Aplicação Gráfica

- Ilustrar uso de reconfiguração dinâmica
- Cada Figura Geométrica é um Componente
- Estado: cor, nome, ID, tamanho
- Três implementações:
 - Quadrado, Círculo, Retângulo Arredondado

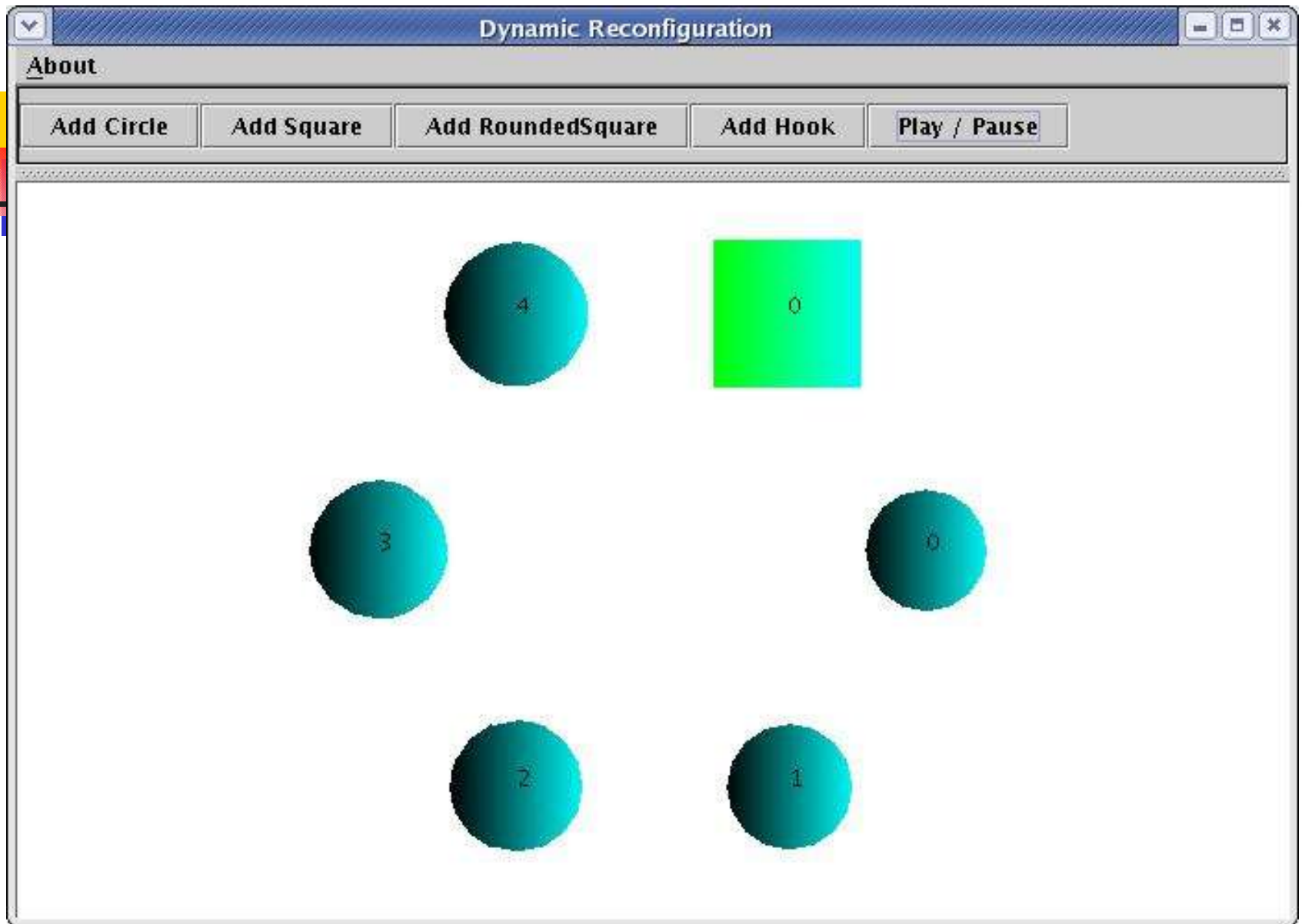
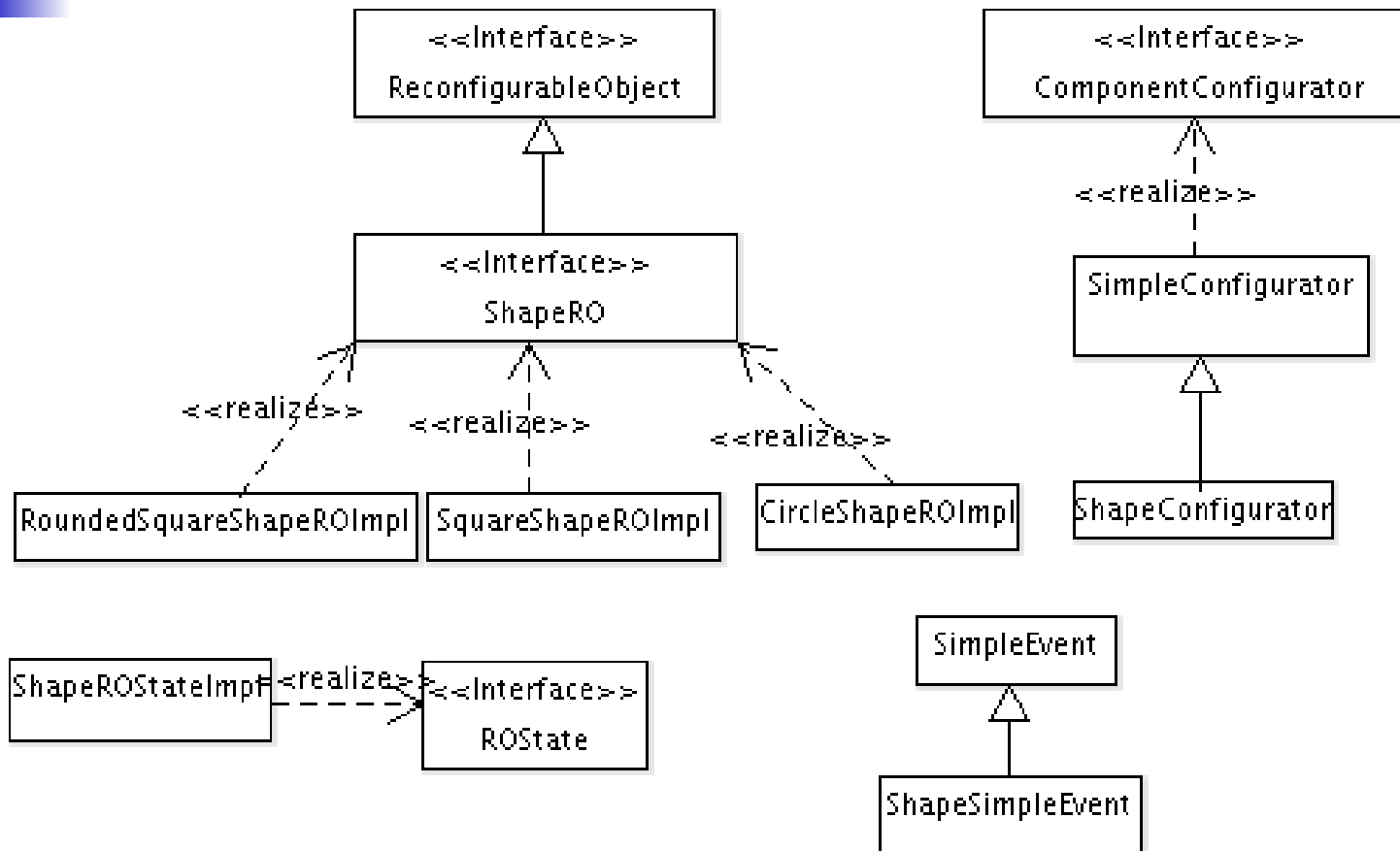
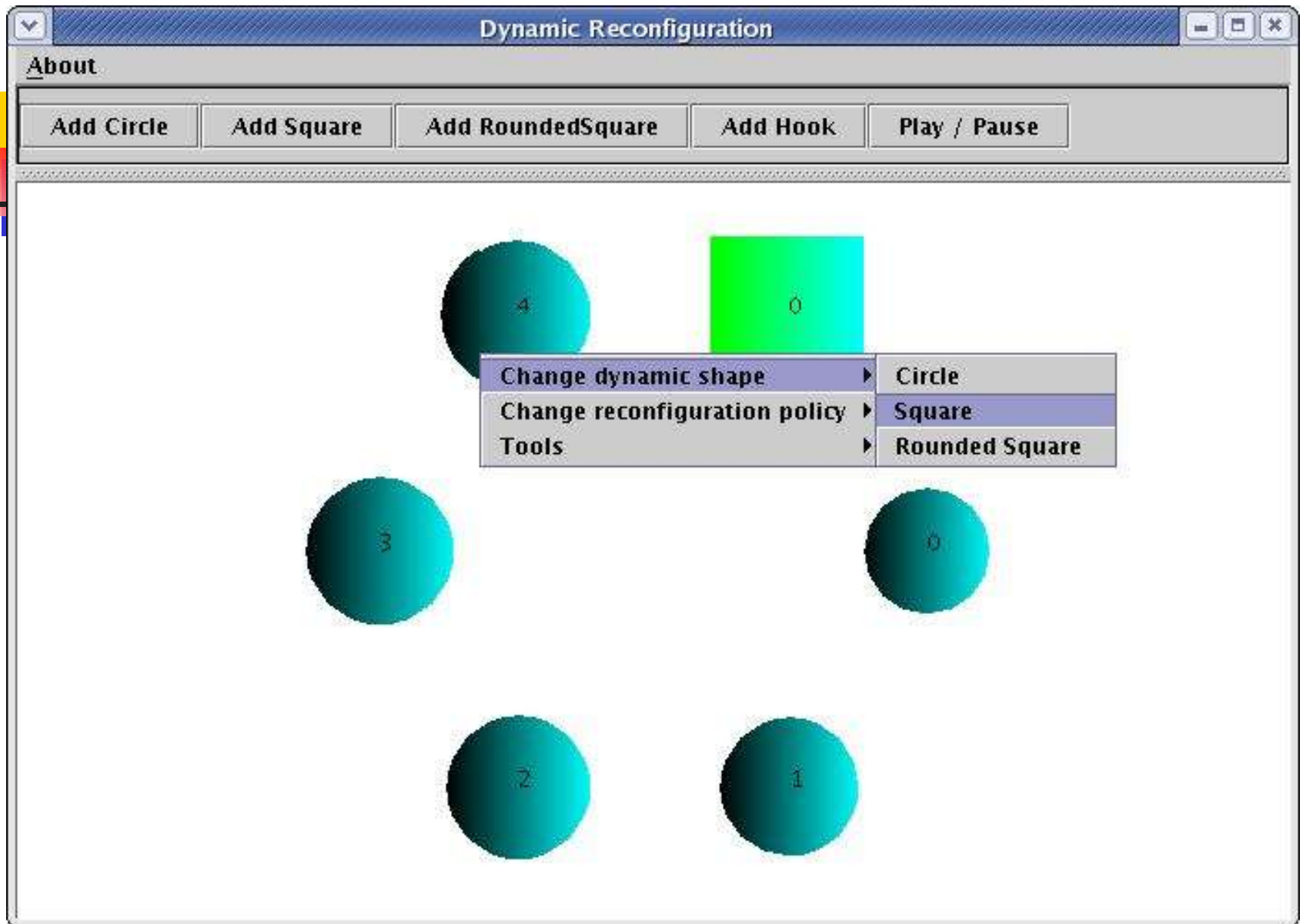
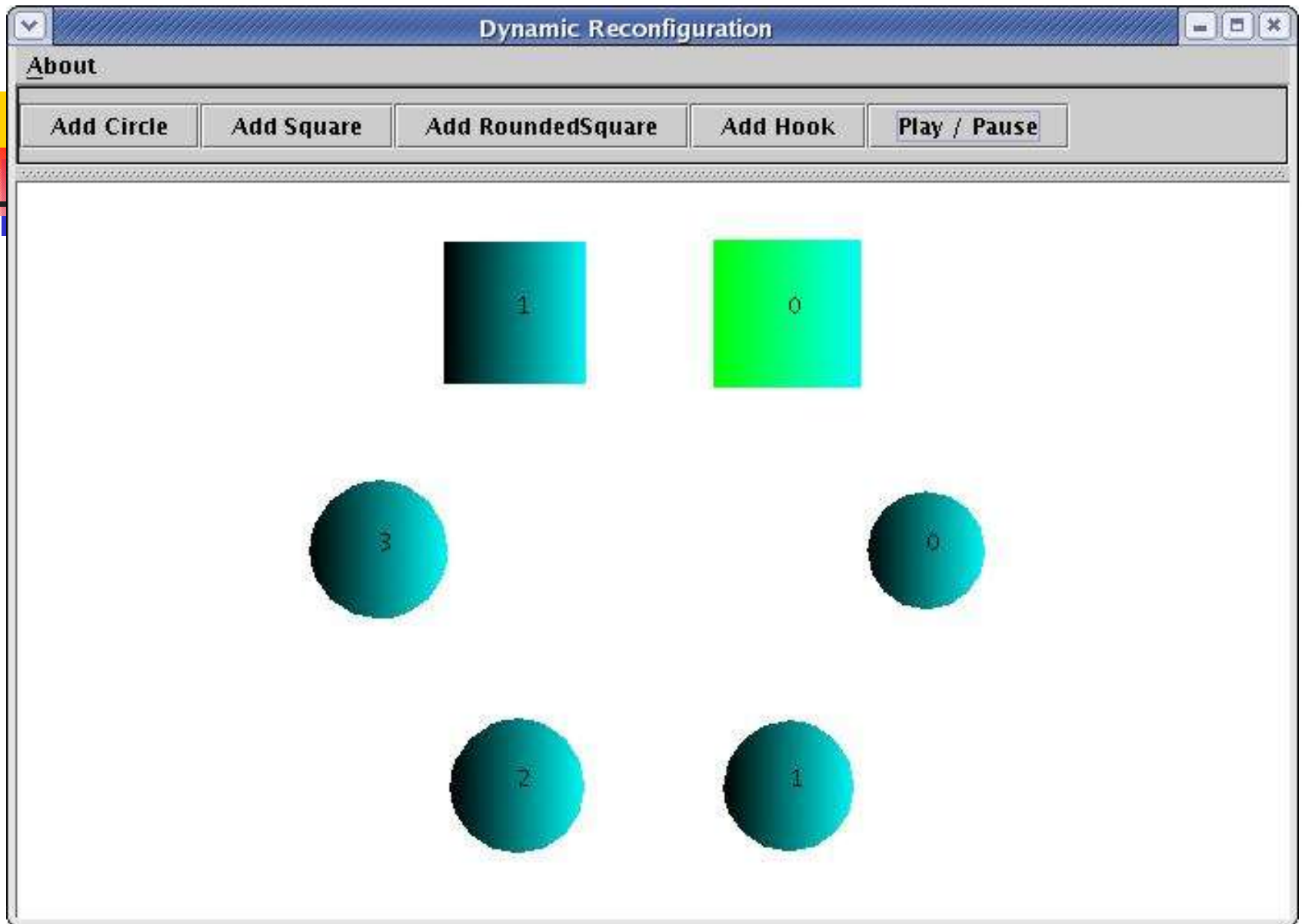
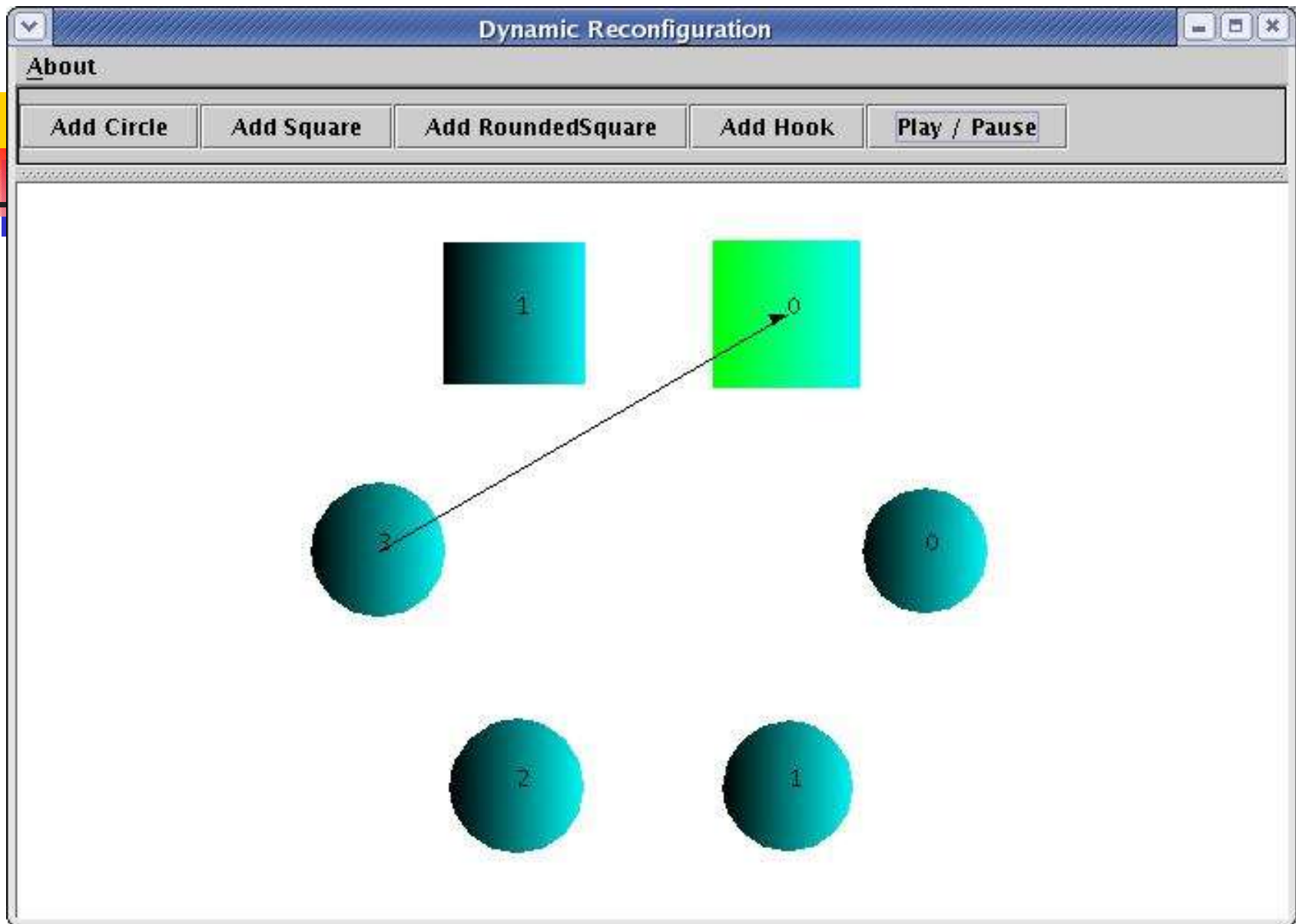


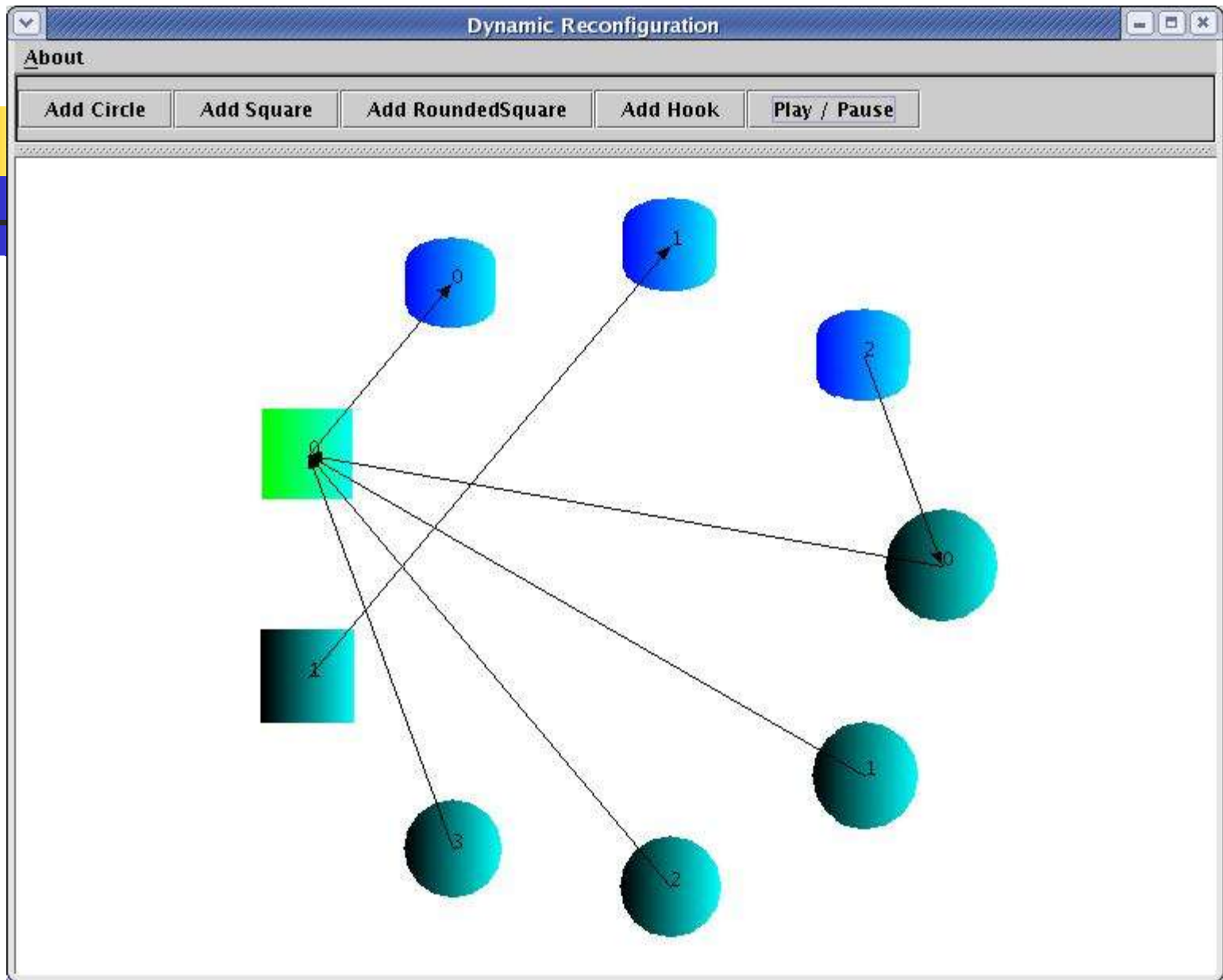
Diagrama de Classes

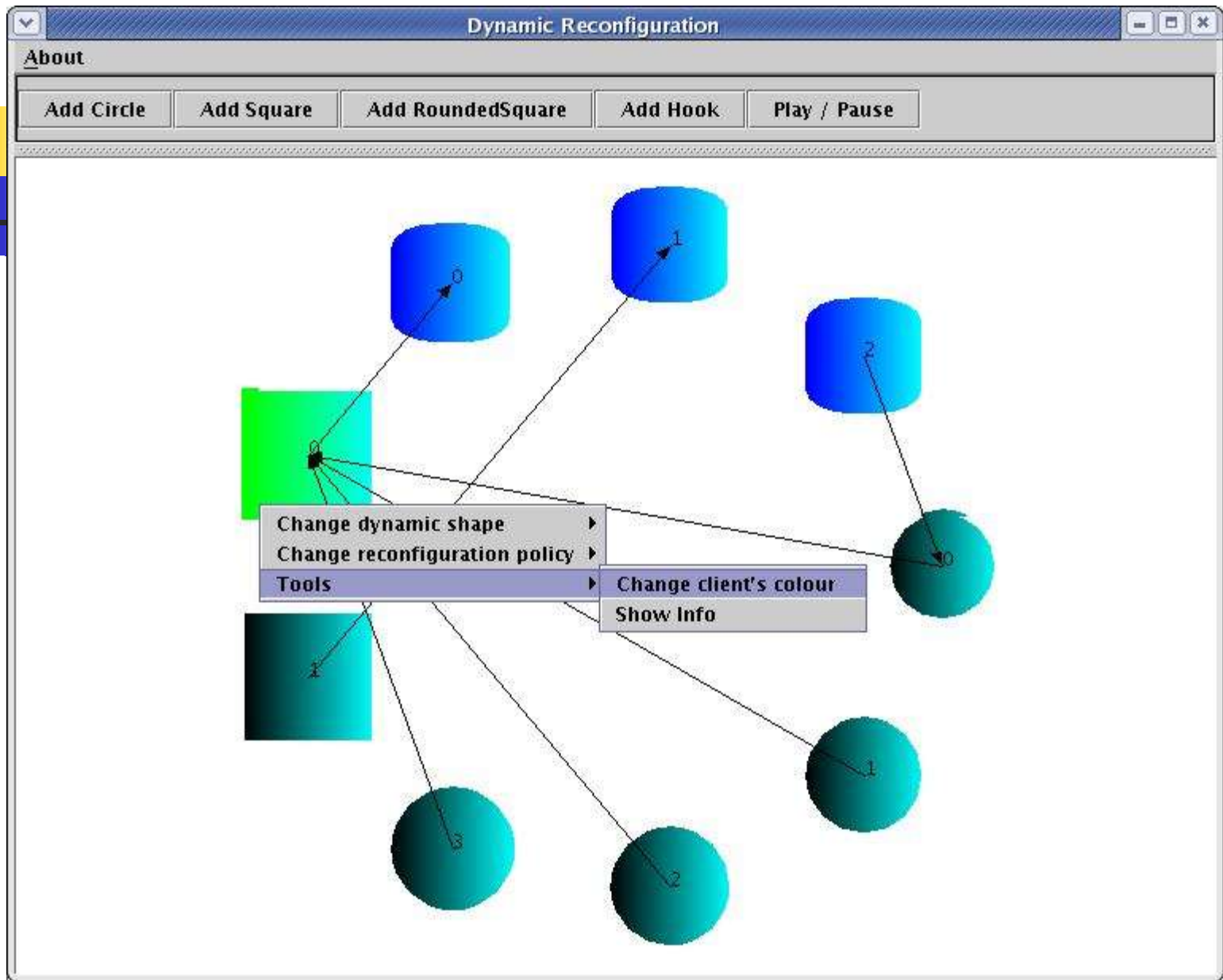


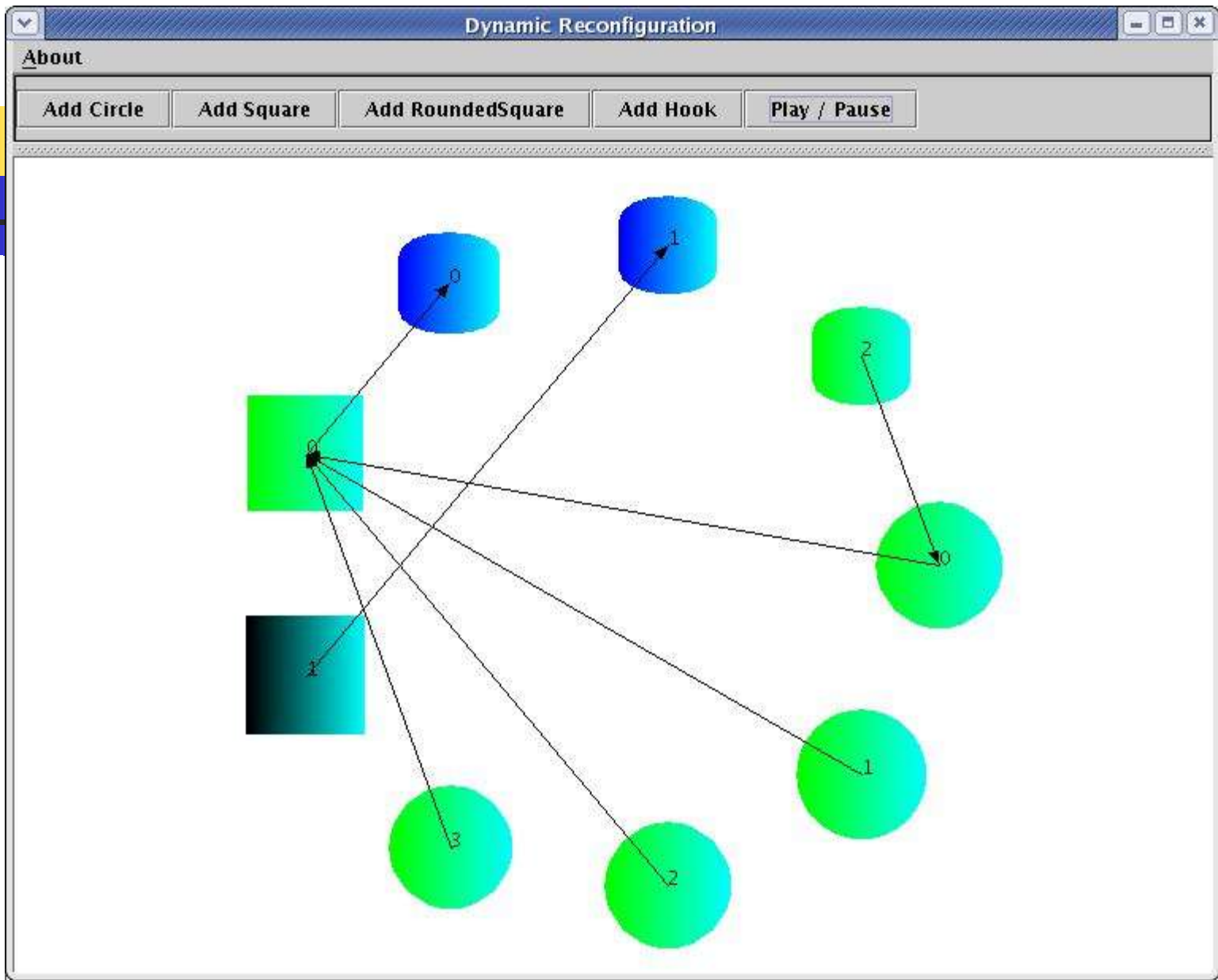














Aplicação Gráfica

- Foi desenvolvida em Java Swing
- A documentação e o código fonte está disponível em:

<http://www.ime.usp.br/~riko>



Conclusões e Trabalhos Futuros

- **Conclusões:**
 - Estudo de mecanismos existentes de reconfiguração
 - Criação de arcabouço para reconfiguração
- **Trabalhos Futuros:**
 - Aspectos transacionais
 - Tolerância a Falhas
 - Suporte a Sistemas Distribuídos



Obrigado!

Agradeço a todos os presentes e a todos
que tornaram esse projeto possível:

OBRIGADO!