Reconfiguração Dinâmica de Componentes em Sistemas Distribuídos de Controle e Supervisão, com Aplicação a Tolerância a Falhas

Neima Prado, Raimundo José de Araújo Macêdo, Luciano Porto Barreto

Laboratório de Sistemas Distribuídos (LaSiD)
Programa de Pós-Graduação em Mecatrônica
Departamento de Ciência da Computação, Universidade Federal da Bahia
Campus de Ondina, CEP: 40170-110, Salvador-BA, Brasil

npsantos@ufba.br, macedo@ufba.br, lportoba@ufba.br

Abstract*. The property of dependability for real-time distributed systems is closely related to its capability to adapt to environmental changes, especially for replacing faulty components. This paper presents the design and implementation of a service that supports dynamic reconfiguration of real-time component-based distributed systems in the control and supervision domain. The service described here was evaluated by the development of a cruise control application in which a faulty controller is replaced by a replica at runtime. The reconfiguration impact on the system performance was analysed by end-to-end control quality metrics and it was considered appropriate to the application being evaluated.*

Resumo*. A qualidade de confiança no funcionamento (dependability) de sistemas de tempo real distribuídos está intimamente ligada à sua capacidade de adaptação às mudanças no ambiente visando, em particular, a reposição de componentes defeituosos. O presente artigo descreve o projeto e implementação de um serviço de reconfiguração dinâmica para o sistema de tempo real distribuído ARCOS - Architecture for Control and Supervision, uma plataforma baseada em componentes de tempo real para aplicações de controle e supervisão. O sistema de reconfiguração dinâmica proposto foi validado através de uma aplicação de controle veicular onde um controlador defeituoso é substituído em tempo de execução por uma réplica. O impacto da reconfiguração, analisada a partir de métricas fim-a-fim de qualidade de controle, se mostrou adequado para a aplicação em questão.*

1. Introdução

A crescente disseminação e complexidade dos sistemas de tempo real demandam a utilização de tecnologias e padrões de projeto que minorem seu custo, tarefas de manutenção e tempo de desenvolvimento, sem negligenciar o atendimento correto de seus requisitos de confiabilidade. Aliado a esse fato, o emprego do *software* no controle de aplicações críticas (*eg*, controle de tráfego, sistemas de energia e telecomunicações) tem evidenciado a necessidade de mecanismos que aprimorem a qualidade de confiança no funcionamento (*dependability*) de tais sistemas, garantindo seu funcionamento correto, mesmo que de forma degradada, durante o maior período de tempo possível, sob pena de acarretar graves prejuízos financeiros ou humanos. A ocorrência de falhas, modificações nas condições do ambiente, obsolescência do software, redução substancial do desempenho

dos processos ou da rede de comunicação entre equipamentos são alguns dos entraves que afetam as aplicações nesse cenário.

Uma forma de contornar a ocorrência de falhas consiste no uso de políticas de reconfiguração dinâmica (em tempo de execução), específicas às aplicações, que lhes permitam adaptação às novas condições do ambiente com a preservação das garantias temporais e funcionais previamente acordadas. A substituição eficiente de elementos faltosos é, portanto, essencial no contexto de sistemas de tempo-real. Além disso, políticas adequadas de reconfiguração possibilitam a concepção de procedimentos de manutenção corretiva e evolutiva que podem melhorar consideravelmente a disponibilidade do sistema. Nesse sentido, o uso da tecnologia de componentes de software tem se mostrado promissora.

Alguns exemplos notórios de tecnologia de componentes incluem Microsoft COM+, .NET [IIOP.Net 2004] e OMG CORBA *Component Model* [Object Management Group 2006]. Como mencionado, as funcionalidades providas por essa tecnologia facilitam a construção de aplicações com requisitos de reconfiguração dinâmica, o que favoreceu também o surgimento de diversos *frameworks* com esse intuito [Batista, Joolia and Coulson 2005, Chan e Wu 2002, Hillman e Warren 2004, Rasche e Polze 2005, Schneider, Picioroagă, e Brinkschulte 2004]. Apesar de tal diversidade, poucos destes *frameworks* de desenvolvimento visam atender às aplicações de controle e supervisão em ambientes industriais de tempo real. Aplicações de controle e supervisão caracterizam-se, principalmente, pela existência de um ciclo de controle, no qual os dados obtidos através do sensoriamento da planta são enviados a um controlador. O controlador, por sua vez, efetua cálculos baseados nos valores medidos e o valor desejado (*setpoint*) das variáveis controladas e determina a ação a ser executada na planta, por meio dos atuadores disponíveis. Tal ciclo de controle gera uma restrição temporal adicional que deve ser satisfeita pelo mecanismo de reconfiguração dinâmica. Outra seja, nesse cenário, é importante que a reconfiguração aconteça de forma a minimizar a interferências danosas ao funcionamento do sistema controlado.

Este trabalho descreve a concepção e implementação de um serviço de reconfiguração dinâmica no contexto de sistemas de tempo real distribuídos. Na abordagem adotada, tanto os elementos constituintes do serviço quanto os da aplicação são implementados como componentes, beneficiando-se de suas características de reusabilidade e encapsulamento inerentes, favorecendo dessa forma a adaptação da aplicação às novas necessidades do ambiente de execução, bem como dos componentes do serviço de reconfiguração. Além disso, a adoção do CIAO (*Component-Integrated ACE ORB*) [Wang 2004], uma adaptação do CORBA *Component Mode*l (CCM) visando atendimento às aplicações distribuídas de tempo real, torna possível ao desenvolvedor da aplicação aproveitar os mecanismos existentes na plataforma utilizada para imprimir maior grau de previsibilidade ao sistema. O serviço apresentado provê primitivas e mecanismos que permitem a implementação de políticas específicas de tolerância a falhas que proporcionam a melhoria da disponibilidade das aplicações. A fim de verificar a adequação do modelo proposto, um protótipo do serviço de reconfiguração foi implementado e avaliado através da análise de métricas de desempenho relativas a uma aplicação de controle veicular, onde um controlador faltoso é substituído por uma réplica usando o mecanismo de reconfiguração dinâmica. Os resultados experimentais obtidos ressaltam a adequação do serviço de reconfiguração à aplicação utilizada.

O serviço de reconfiguração proposto foi concebido para ser integrado ao *framework* ARCOS (ARquitetura de COntrole e Supervisão) [Andrade e Macêdo 2005, Andrade et al 2006, Andrade e Macedo 2007, Macêdo et al 2004] cujo objetivo é prover uma plataforma reutilizável e extensível para sistemas de supervisão e controle, abordando aspectos como interoperabilidade e mecanismos para garantias temporais e cuja plataforma também utiliza o CIAO.

O restante desse artigo está estruturado da seguinte maneira. A Seção 2 fornece uma visão geral do serviço de reconfiguração de componentes. A Seção 3 apresenta uma aplicação-exemplo e aspectos de sua implementação no serviço de reconfiguração proposto. A Seção 4 descreve a avaliação experimental da aplicação-exemplo através de alguns cenários de reconfiguração. Os principais trabalhos relacionados ao tema deste artigo são discutidos na Seção 5 e, por fim, a Seção 6 conclui o artigo apresentando considerações finais e perspectivas de trabalhos futuros.

2. Serviço de reconfiguração dinâmica de componentes

O serviço de reconfiguração dinâmica, proposto neste trabalho, objetiva prover mecanismos que permitam modificar a configuração do sistema em tempo de execução, preservando a consistência. Nas seções seguintes, apresentaremos o modelo do sistema e uma visão geral sobre a arquitetura do serviço de reconfiguração proposto.

2.1 Modelo do sistema

O sistema é formado por componentes conectados através de portas. Através das portas, o componente especifica quais os serviços que provê aos demais, bem como suas dependências em relação aos outros componentes. A interação dos componentes é feita através da conexão de portas de provisão, denominadas facetas, com portas de recepção de serviços, denominadas receptáculos. Para uso dos serviços providos por outro componente, é preciso haver uma conexão entre o receptáculo do componente e a faceta do componente provedor. Cada tipo de componente é gerenciado por uma implementação da interface *Home*, que provê operações para criação e localização de componentes. Assume-se que as falhas dos componentes são do tipo *fail-stop*.

2.2 Arquitetura do serviço de reconfiguração

A arquitetura proposta do serviço de reconfiguração é dividida em três elementos: *Reconfiguration Manager* (RM), *Consistency Manager* (CM), e *Deployment Manager* (DM). O *Reconfiguration Manager* é o principal responsável e coordenador do procedimento de reconfiguração. Para efetuar uma política de reconfiguração, este utiliza o *Consistency Manager* e o *Deployment Manager*. O primeiro tem por objetivo preservar a consistência global do estado do sistema, em especial, conduzindo os componentes a um estado consistente através do (des)bloqueio controlado de suas conexões e de sua (re)inicialização. O *Deployment Manager*, por sua vez, é utilizado para criar e destruir componentes ou as conexões entre componentes. A Figura 1 ilustra uma inter-relação simplificada entre estes elementos, descritos nas seções seguintes.

2.2.1 Reconfiguration Manager (RM)

O RM é o componente responsável por gerenciar a reconfiguração do sistema. Através deste, é possível criar e remover componentes e conexões, além de transferir estado entre componentes. As operações de reconfiguração possíveis através do serviço de

reconfiguração são: criação e remoção de componentes, transferência de estado entre eles e (re)conexão de suas portas. É possível submeter cada operação de reconfiguração separadamente ou através de um *script* de reconfiguração com operações no formato XML.

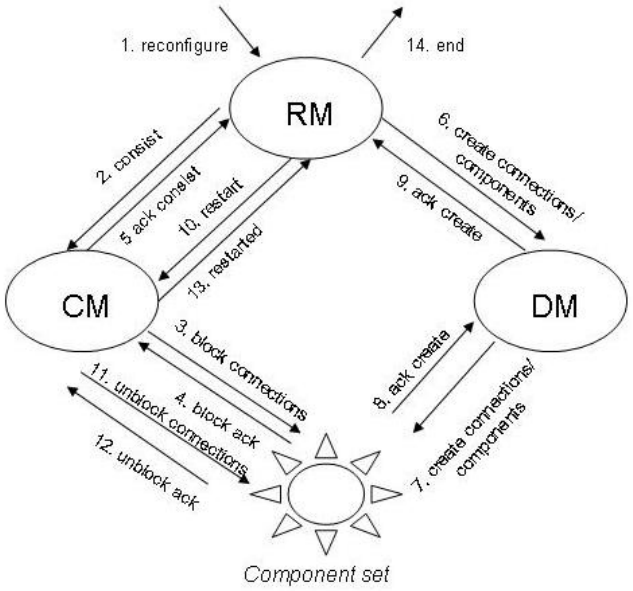

Figura 1. Visão geral do serviço de reconfiguração

Conforme ilustrado na Figura 1, a reconfiguração inicia através da ativação do método *reconfigure*, que recebe como parâmetro um conjunto de operações de reconfiguração. Caso a aplicação tenha definido uma política de consistência através de um CM, este é utilizado para conduzir o sistema a um estado adequado para a reconfiguração (Seção 2.2.2). Por exemplo, no cenário explorado na seção 3 (controle de velocidade veicular), considera-se que o estado de consistência foi alcançado através da abordagem sugerida em [Wermelinger 1999]. Essa abordagem bloqueia somente as conexões necessárias à reconfiguração, o que favorece a continuidade de outros serviços providos pelo componente, aumentando a disponibilidade da aplicação. Em seguida, o RM se conecta ao DM para acionar as mudanças estruturais da aplicação, descritas na Seção 2.2.3.

O RM permite a criação de políticas de reconfiguração através da definição de eventos e ações correspondentes de reconfiguração, que são acionadas quando da ocorrência de eventos. Para facilitar a escrita de políticas de reconfiguração nesse ambiente, foi concebida uma linguagem específica que permite descrever cenários de reconfiguração de forma mais simples e clara do que em XML. A linguagem é baseada no paradigma de comandos guardados, no qual a ocorrência de eventos ou avaliação verdadeira de predicados dispara a execução de blocos de comandos. Um exemplo de cenário de reconfiguração é descrito na Seção 3.1.

2.2.2 Consistency Manager (CM)

O CM é o componente responsável por conduzir o sistema a um estado apropriado para a realização da reconfiguração. Fundamentalmente, sua atuação concerne o envio de ordens (implementadas através de eventos) aos componentes da aplicação participantes do procedimento de reconfiguração. Para isso, os componentes envolvidos na ação de

reconfiguração devem utilizar portas receptoras de eventos específicos para este fim. Estas ordens envolvem o bloqueio e desbloqueio das conexões destes componentes.

Para alcançar um estado consistente de reconfiguração, o CM mantém o registro das conexões da aplicação e se comunica com os componentes de forma assíncrona, solicitando o bloqueio de conexões, quando devido. Como ilustra a Figura 1, após ser ativado pelo RM (*2. consist*), o CM efetua o bloqueio das conexões envolvidas (*3. block connections*). Por sua vez, o componente informa ao CM quando a solicitação de bloqueio for atendida (*4. block ack*). Para evitar *deadlock*, as solicitações de bloqueios e desbloqueios consideram a dependência entre as conexões; ou seja, uma conexão é bloqueada somente quando as conexões que dela dependem já estiverem bloqueadas. Do mesmo modo, uma conexão é desbloqueada somente quando as conexões das quais depende já o tiverem sido.

Uma vez que todas as solicitações de bloqueio tenham sido atendidas, o CM informa a situação ao RM (*5. ack*), que prossegue com a reconfiguração interagindo com o DM (Seção 2.2.3). Finda esta interação, o RM solicita ao CM que desbloqueie as conexões (*10. restart*) que, por sua vez, notifica os componentes a esse respeito (*11. unblock connections*) de modo que o sistema possa retornar ao seu estado normal de funcionamento.

A depender da abordagem de consistência adotada, o CM pode apresentar comportamentos distintos. Por exemplo, utilizando-se a abordagem de preservação de consistência descrita por [Kramer and Magee 1990], todos os nós que participam direta ou indiretamente de conexões a serem removidas devem ser conduzidos ao estado passivo, no qual apenas atendem requisições de cuja execução dependa a completude de outras. Já na abordagem orientada a conexões de Wermelinger [Wermelinger 1999], o estado para reconfiguração é alcançado através do bloqueio das conexões que serão removidas, obedecendo-se uma ordem dada pela dependência entre elas. Observa-se, portanto, que a determinação dos elementos a terem seu comportamento alterado e a ordem em que isso é realizado diverge a depender da estratégia empregada. Através da arquitetura apresentada no presente trabalho, é possível implementar *Consistency Managers* com comportamentos distintos e conectá-los ao *Reconfiguration Manager*, *Deployment Manager* e componentes da aplicação. Obviamente, estes últimos devem estar preparados para interagir com o CM através da provisão de portas de recepção e de emissão de eventos adequados à estratégia adotada.

2.2.3 Deployment Manager (DM)

O *Deployment Manager* é responsável pela criação, remoção de conexões e transferência de estado entre componentes. Estas funcionalidades são disponibilizadas através de uma interface que provê, entre outras, as seguintes operações: *Create/Remove Component*, *Create/Remove Connection* e *Transfer state*. A operação *Create component* recebe como parâmetro um identificador único para o componente, o nó em que o componente deve ser instanciado e o tipo do componente. A remoção do componente é feita através do fornecimento do seu identificador. As conexões são criadas através do comando *Create connection*, para o qual deve ser informado: o identificador da conexão e os componentes com respectivas facetas e receptáculos a serem conectados. A remoção de uma conexão requer apenas o identificador da conexão. Por fim, a operação *Transfer state*: efetua a transferência de estado entre componentes de mesmo tipo. Para isso, devem ser informados os identificadores dos componentes fonte e destino.

3. Estudo de caso: Reconfiguração dinâmica de uma aplicação de controle de velocidade veicular para substituição de um controlador falho

Para validação da abordagem, utilizamos o serviço de reconfiguração para gerenciar a falha de um dos componentes de uma aplicação de controle veicular. Nesse cenário, a detecção da falha de um controlador dispara o procedimento de substituição desse componente por uma réplica equivalente. Este serviço foi implementado no CIAO [Wang 2004], mesmo *middleware* com enfoque em aplicações distribuídas de tempo-real utilizado para a implementação do ARCOS. A seguir detalhamos as características principais dessa aplicação.

Na aplicação de controle veicular, o motorista informa a velocidade em que deseja trafegar e o sistema controla a atuação necessária a ser efetuada no acelerador do veículo para atingir esse objetivo. O comportamento físico da simulação do veículo é caracterizado pelas equações [Pont 2001]:

$$Accel = (Throttle * ENGINE\_POWER – (FRIC * Old\_speed)) / MASS \quad (1)$$

$$Dist = Old\_speed + Accel * (1/SAMPLE\_RATE) \quad (2)$$

$$Speed = SQRT ((Old\_speed)^2 + 2 * Accel * Dist) \quad (3)$$

O veículo apresenta propriedades de potência do motor, coeficiente de fricção e massa que são representadas pelas constantes ENGINE_POWER, FRIC e MASS. Informados a velocidade corrente (*Old_speed*) e o valor da atuação no acelerador (*Throttle*), se obtém a velocidade do veículo (*Speed*). A taxa de amostragem utilizada pelo controlador é representada pela variável SAMPLE_RATE.

Esse modelo serviu de base para implementação de um controlador PID (Proporcional-Integrativo-Derivativo) [Ogata 2001] que atua diretamente no acelerador com base no erro mensurado entre a velocidade atual e a velocidade desejada (*setpoint*). A calibração do valor calculado na saída do controlador depende da ponderação de três parâmetros essenciais: ganho proporcional (*kp*), ganho integral (*ki*) e ganho derivativo (*kd*). O termo proporcional enfatiza a reação do controlador aos erros momentâneos. O termo integral considera a reação segundo o somatório dos erros (histórico). Por fim, o termo derivativo determina a reação em função da taxa de modificação do erro mensurado (derivada). A sintonia desses parâmetros visa atender aos requisitos da aplicação tais como tempo de subida, tempo de estabilização e *overshoot* [Ogata 2001].

A implementação dos componentes dessa aplicação está apresentada na Figura 2. O *Gerador de Pulso* produz pulsos, como referências temporais, para o *Sensor* de velocidade na taxa de amostragem especificada pela aplicação de controle, cujo valor padrão é de 2 Hz. A cada pulso recebido, o *Sensor* obtém o valor da velocidade atual e a repassa para o *Controlador1*. Este, por sua vez, calcula o novo valor de atuação no acelerador de acordo com o erro entre o valor medido e o valor desejado para a velocidade.

A falha do *Controlador1* é tolerada através de uma variante do esquema de replicação semi-ativa, sendo que o estado da réplica (*Controlador2*) é atualizado a cada novo cálculo do valor de atuação. O *Controlador1* repassa os valores de atuação ao *Atuador*, que atualiza os parâmetros correspondentes no objeto controlado, representado pelo componente *Veículo*. A detecção de falhas é implementada através da conexão utilizada para transferência de estado existente entre o *Controlador2* e o *Controlador1*. Caso o intervalo entre duas operações de atualização de estado ultrapasse determinado

valor especificado no *Controlador2*, este produz um evento de notificação de falha ao *Reconfiguration Manager*. Este último, por sua vez, aciona a reconfiguração do sistema, de acordo com a política especificada. Assume-se, que na ocorrência de falhas, o *Controlador1* possui um comportamento do tipo *fail-stop*.

As equações (1), (2) e (3) são utilizadas pelo *Sensor* para calcular a velocidade atualizada do *Veículo*. Desse modo, ao receber o pulso do *Gerador de Pulso*, o *Sensor* se comunica com o *Veículo* obtendo o valor atual da variável *throttle*. Este valor é substituído na equação (1) para calcular a aceleração do veículo, que é dependente da atuação no acelerador e da velocidade anterior. Conhecendo-se a aceleração, a distância que o veículo percorreu durante o tempo transcorrido entre uma medição e outra é obtida através da equação (2). Finalmente, a equação (3) é empregada para o cálculo da velocidade atual do *Veículo*. Este valor é o que é enviado ao *Controlador1* para ser comparado com o *setpoint*, para correção da atuação, caso necessário.

A estratégia de preservação de consistência implementada foi a de bloqueio de conexões [Wermelinger 1999], mencionada na seção 2.2.2.

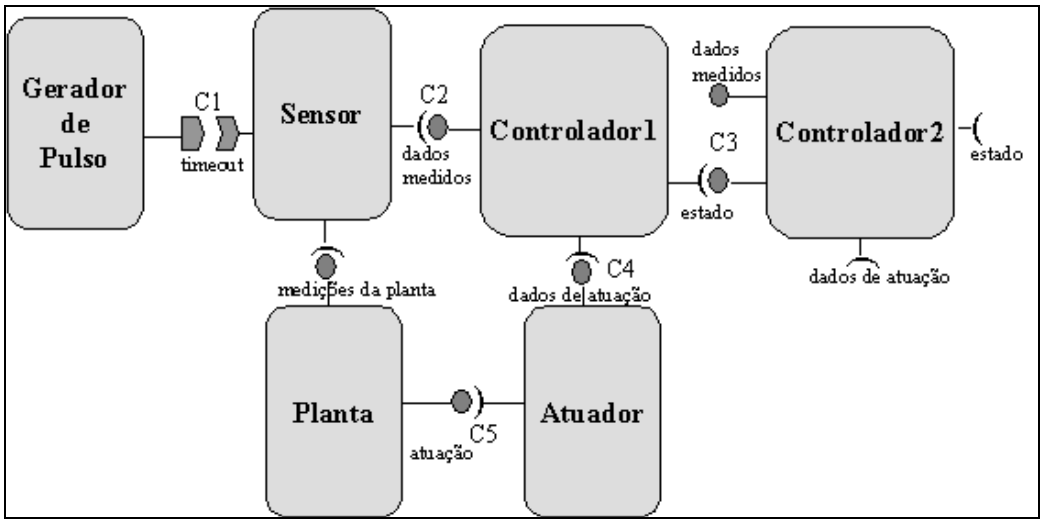

Figura 2. Componentes da aplicação de controle de velocidade veicular

3.1 Exemplo de cenário de reconfiguração

Esta seção apresenta um cenário de reconfiguração que efetua a substituição de um controlador falho através da reorganização das conexões das quais ele participa. A política de reconfiguração é escrita numa linguagem específica que fornece abstrações de alto nível para o desenvolvedor da política de reconfiguração e permite a geração automática de código para XML. A política de reconfiguração de substituição de um controlador falho é apresentada no trecho de código da Figura 3.

```
01 (Controlador1.failed) => {
02    destroy connection C2, C3, C4
03    create connection C6 : Sensor.r1 -> Controlador2.f1
04    create connection C7 : Controlador2.r1 ->  Atuador.f1  }
```

Figura 3. Política de reconfiguração de um controlador falho

A notificação da falha do componente *Controlador1* ao *Reconfiguration Manager*, expressa através do evento `Controlador1.failed`, aciona a remoção das conexões originais e efetua a conexão do novo controlador aos componentes existentes. Em primeiro lugar, o *script* remove as conexões originais (*C2, C3* e *C4*) através do comando `destroy` (linha 2), que atua sobre uma lista de conexões existentes. Nas linhas 03 e 04 são estabelecidas duas conexões (*C6* e *C7*) entre o novo controlador e os componentes pertinentes (*Sensor* e *Atuador*). Os descritores f1 e r1 representam as facetas e receptáculos dos componentes envolvidos no procedimento de conexão. A título de exemplo, o trecho de código da Figura 4 representa o código XML gerado para o comando de criação da conexão *C6* entre o componente *Controlador2* e o Sensor (linha 3) da Figura 3.

```
01 <command>
02   <createConnection>
03     <id>C6</id>
04     <type>SIMPLEXRECEPTACLE_FACET</type>
05       <initiator>
06         <portName>r1</portName>
07         <instance>Sensor</instance>
08       </initiator>
09       <receptor>
10         <portName>f1</portName>
11         <instance>Controlador2</instance>
12       </receptor>
13   </createConnection>
14 </command>
```

Figura 4. Código XML referente à criação do Controlador3 (linha 3 da Figura 3)

A utilização de uma linguagem específica facilita a verificação automática de propriedades importantes no contexto do domínio de reconfiguração. Por exemplo, no cenário de falha exposto, espera-se que o comando `destroy` atue somente nas conexões nas quais haja participação do componente falho (*C2*, *C3* e *C4*, no caso), pois não faz sentido, além de ser extremamente indesejável, afetar as conexões de outros componentes nesse contexto. Tal verificação é facilmente realizável através de análise simples dos comandos do *script* de reconfiguração.

4. Avaliação de desempenho

O experimento foi conduzido em um computador com processador Pentium IV 2.66GHZ, 512MB de memória RAM e sistema operacional Debian Linux kernel 2.6. Os valores das propriedades do veículo e de configuração do controlador foram extraídos de [Andrade 2006]: ENGINE_POWER=5000, FRIC=50, SAMPLE_RATE=2Hz, termo proporcional do controlador PID = 0.05, Termo derivativo do controlador PID = 0, termo integral do controlador PID = 0. Nos experimentos efetuados, o *script* de reconfiguração submetido ao *Reconfiguration Manager* continha os seguintes comandos: remover as conexões *C2*, *C3* e *C4*, remover o controlador original, criar conexão *C6* entre o novo controlador e o sensor e, por fim, criar conexão *C7* entre o atuador e o novo controlador. Os tempos obtidos são relativos ao início da reconfiguração pelo RM, após a falha ter sido notificada pelo *Controlador2*.

4.1 Métricas de desempenho

Para a aplicação veicular, o aspecto principal na avaliação do controlador consiste em verificar sua capacidade em manter a velocidade próxima ao valor desejado. Para este fim,

três métricas de avaliação foram consideradas [Ogata 2001]: tempo de subida, tempo de estabilização e *overshooting*. O tempo de subida (*raise time*) é o tempo necessário para o valor atual (resposta) alcançar uma faixa percentual do valor desejado. Três faixas de valores são normalmente utilizadas: 10% a 90%, 5% a 95% e 0% a 100%. O tempo de estabilização (*settling time*) é o tempo necessário para a curva de resposta alcançar e permanecer dentro de um intervalo em torno do valor final. Este intervalo é especificado por uma porcentagem absoluta do valor final (geralmente entre 2% a 5%). A última métrica é o percentual máximo de *overshoot* que corresponde ao valor máximo obtido pela curva de resposta.

Para avaliar o impacto da reconfiguração no desempenho da aplicação durante a reconfiguração foram definidos três cenários: reconfiguração simples; reconfiguração com modificação do *setpoint* e reconfiguração com simulação de perturbação, tendo sido a reconfiguração acionada após a estabilização da velocidade. Para cada cenário, foram obtidas as curvas de resposta da velocidade do veículo, com e sem o procedimento de reconfiguração, de modo a avaliar o impacto da reconfiguração no desempenho da aplicação de controle. Estes cenários de avaliação são descritos nas seções seguintes.

4.2 Cenário 1: Reconfiguração simples

Este cenário consiste na aplicação de reconfiguração (através da falha forçada do *Controlador1*) após a estabilização do valor da velocidade do veículo em 40 km/h. A Figura 6 descreve a curva de evolução da velocidade do veículo na execução de reconfiguração aos 59s; portanto, após atingida a estabilidade da velocidade. A proximidade das curvas (a diferença é praticamente imperceptível no gráfico) de evolução da velocidade nas situações com a reconfiguração e sem a reconfiguração atesta a ausência de impacto significativo na aplicação.

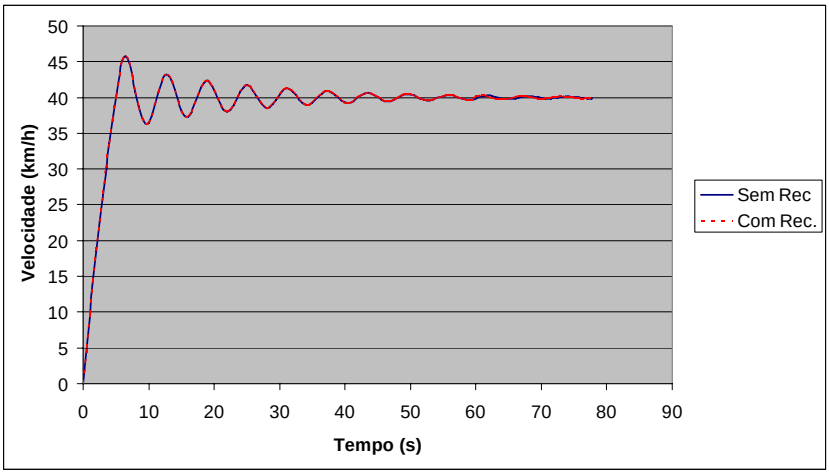

Figura 6. Evolução do Controlador PID de velocidade com reconfiguração aos 59s

Observaram-se os mesmos valores para as propriedades do sistema controlado com e sem a reconfiguração: tempo de subida ≈ 5 s, tempo de estabilização ≈ 22 s (considerando uma variação em torno de 5% do valor final) e *overshoot* ≈ 14 %. Os custos referentes às atividades de reconfiguração são: intervalo entre a solicitação de bloqueio da conexão e seu bloqueio efetivo 460 ms, o intervalo entre a solicitação de desbloqueio da conexão e seu desbloqueio efetivo: ≈ 500 ms.

4.3 Cenário 2: Reconfiguração com modificação de *setpoint*

Este cenário consiste na aplicação de reconfiguração após a estabilização do valor da velocidade do veículo em 40 km/h, e posterior mudança de *setpoint* para 70 km/h, seguida de reconfiguração aos 100 s. Assim como no cenário anterior, a Figura 7 ilustra diferença desprezível entre os tempos na ausência e presença da reconfiguração.

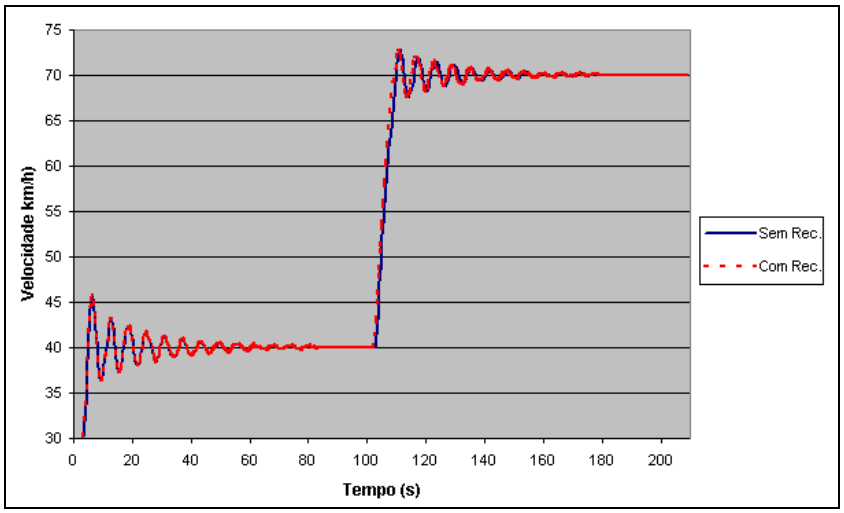

Figura 7. Evolução do Controlador PID com alteração da velocidade de 40km/h para 70km/h aos 100 segundos

A Tabela 1 apresenta os valores do tempo de subida, tempo de estabilização e *overshoot* quando da modificação do *setpoint*. Vale ressaltar que os valores para o *setpoint* em 40 km/h são os mesmos do ambiente de testes de reconfiguração simples (cenário 1).

Setpoint	Tempo de subida	Tempo de estabilização	*Overshoot*
40 km/h	5s	22s	14%
70 km/h	10s	9s	4%

Tabela 1. Métricas de desempenho do controlador PID com modificação de *setpoint*

4.4 Cenário 3: Reconfiguração perturbação simulada

Este cenário consiste na aplicação de reconfiguração após a estabilização do valor da velocidade do veículo em 40 km/h e posterior perturbação ao veículo através da simulação de rajada de vento em duas situações distintas. Na primeira situação, o valor da variável *throttle* foi alterado para 1.5 no instante 100s, simulando a ocorrência de vento incidente na traseira do veículo, ilustrado pela Figura 9. Logo após a perturbação, foi submetido o *script* de reconfiguração. Nesse caso, o sistema leva aproximadamente 2 s para alcançar o *setpoint*, ou seja, tanto com e sem reconfiguração, o tempo de subida é de aproximadamente 2 s. Em ambos os casos, o tempo de estabilização após a perturbação é de por volta de 4s e o *overshoot* é de aproximadamente 14%.

Na segunda situação, no instante 30 s, simulou-se a incidência de vento na dianteira do veículo modificando bruscamente o valor da variável *throttle* para 0,1. Após a perturbação (seguida imediatamente pela reconfiguração), o tempo de subida teve o valor de aproximadamente de 0,4 s. Observou-se que, após a perturbação, em momento algum a variação da velocidade alcançou valor inferior ou superior a 5% do *setpoint*. O *overshoot* ficou em torno de 4 %.

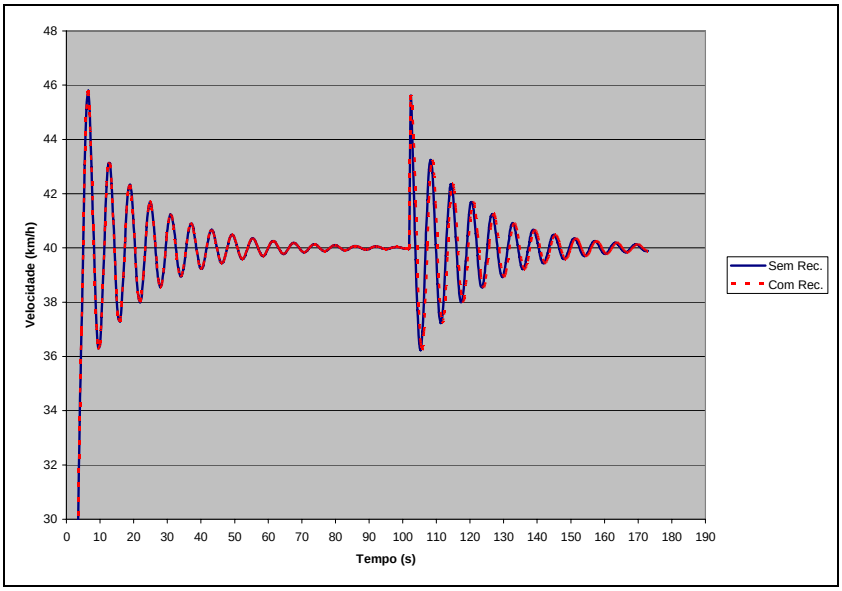

Figura 9. Evolução do Controlador PID com simulação de rajada de vento na traseira do veículo no instante 100 s

4.5 Discussão

Com base nos experimentos apresentados, observa-se que as ações de reconfiguração não tiveram impacto significativo no desempenho do controle do veículo. No entanto, a reconfiguração influi no comportamento do sistema, uma vez que tem como conseqüência o bloqueio das conexões entre o *Sensor* e o *Controlador* e entre o *Controlador* e *Atuador*. Por esse motivo, durante o tempo de reconfiguração, o *Sensor* não atualiza o *Controlador* com a velocidade do veículo, fazendo com que o valor da atuação permaneça o mesmo nesse período.

No experimento 1, uma vez que a reconfiguração se dá em um momento de estabilidade do sistema, sem influência de eventos externos, era esperado que ela tivesse pouco impacto no comportamento do mesmo. Nos cenários 2 e 3, a reconfiguração foi acionada imediatamente após a alteração do *setpoint* e da ocorrência de perturbação, o que poderia fazer com que a resposta do sistema a esses eventos fosse prejudicada pelo atraso causado pela reconfiguração. Entretanto, conforme observado nos gráficos e nas métricas do sistema, não houve influência significativa do tempo de reconfiguração no desempenho do mesmo. Tal comportamento pode ser atribuído ao fato de o intervalo de tempo em que as conexões do sistema permanecem bloqueadas ser em torno de 100 ms. Além disso, a taxa de amostragem variou entre 100 e 500 ms, o que possibilita ao *Controlador* corrigir a velocidade do *Veículo*, no intervalo de tempo relativamente curto. Espera-se que em aplicações em que seja necessário efetuar reconfigurações mais demoradas (por exemplo, com mais operações de re-conexão), a resposta do sistema a uma alteração de *setpoint* ou distúrbio externo seja mais prejudicada do que o observado nos experimentos apresentados.

5. Trabalhos Relacionados

Diversos trabalhos anteriores versaram sobre a reconfiguração dinâmica de componentes. Relacionamos, a seguir, alguns dos trabalhos relevantes na área.

Bidan et al (1998) propõem um serviço de reconfiguração dinâmica baseado em CORBA que permite criar e remover objetos e conexões, desde que inexistam chamadas de

RPC pendentes entre os objetos envolvidos nas conexões. Em [Chan e Wu 2002] é definido um modelo de componentes denominado ComponentGOP, implementado sobre CORBA, cuja arquitetura é representada por grafos e a reconfiguração é dirigida por primitivas que atualizam os grafos. OpenRec [Hillman e Warren 2004] é um *framework* de reconfiguração dinâmica que utiliza um modelo próprio de componentes e disponibiliza um algoritmo básico de reconfiguração que deve ser especializado para atender aos requisitos de consistência da aplicação. Em [Rasche and Polze 2005] é apresentada uma infra-estrutura de reconfiguração em plataformas .NET que permite a criação, remoção e migração de componentes e utiliza a abordagem de Wermelinger [Wermelinger 1999].

LuaSpace é um ambiente para reconfiguração dinâmica de aplicações baseadas em CORBA [Batista, Cerqueira e Rodriguez 2003], no qual os objetos são acessados através de *proxies* que intermedeiam sua comunicação com o cliente. A depender da configuração da aplicação, o *proxy* encaminha a requisição ao objeto adequado através da utilização dos mecanismos de Repositório de Interfaces e Interface de Invocação Dinâmica do CORBA.

O SwapCIAO [Balasubramanian et al 2005] é uma extensão do *middleware* CIAO [Wang 2004] que permite atualizar a implementação de um componente em tempo de execução. Para tanto, dispõe de mecanismos para redirecionar os clientes de um componente existente para uma nova instância do mesmo. Plastik [Batista, Joolia and Coulson 2005] é um *meta-framework* que provê reconfiguração dinâmica através da combinação de uma linguagem de descrição arquitetural (ADL) com um modelo de componentes reflexivo chamado OpenCOM. Plastik fornece dois tipos de reconfiguração: programada e *ad-hoc*. A reconfiguração programada é feita através da especificação de comandos do tipo "predicado-ação", de modo similar ao nosso trabalho, ao passo que a reconfiguração *ad hoc* pode ser realizada através de *scripts* em ADL ou de comandos do OpenCOM.

Em [Hofmeister and Purtilo 1993] a aplicação é estruturada na forma de módulos nos quais podem ser definidos pontos de reconfiguração. Tais pontos indicam quando a reconfiguração é segura e o estado da aplicação. O código fonte do programa é transformado através da adição dos comandos necessários para: (i) postergar a reconfiguração até o momento apropriado, (ii) organizar e enviar o estado, (iii) instalar o estado, restaurar a pilha de registro de ativação quando necessário, (iv) reiniciar a execução no lugar apropriado. Em [Schneider, Picioroagă, and Brinkschulte 2004] é definido um serviço de reconfiguração integrado ao middleware OSA+ (*Open System Architecture*). Os serviços se comunicam através de tarefas (*jobs*) , onde uma tarefa é um par formado por uma ordem e um resultado e a reconfiguração é efetuada através da substituição ou movimentação de serviços, com e sem transferência de estado.

Apesar de disporem de ações de reconfiguração similares às apresentadas neste trabalho, em [Bidan et al 1998] o modelo de sistema empregado é o de objetos definido pelo CORBA. No ComponentGOP [Chan e Wu 2002] e no OpenRec [Hilman e Warren 2004] utiliza-se um modelo de componentes próprio, sendo que o do primeiro é baseado em CORBA. Embora o Plastik [Batista, Joolia and Coulson 2005] seja baseado em um modelo de componentes, seu foco não visa aplicações com necessidade de garantias temporais. O trabalho de [Schneider, Picioroagă, and Brinkschulte 2004] tem por base um middleware de tempo-real, entretanto, o sistema não utiliza um modelo de componentes e, portanto, se limita à substituição de serviços.

O modelo de componentes e middleware deste trabalho são os mesmos empregados no SwapCIAO [Balasubramanian et al 2005]. Entretanto, as funcionalidades previstas no SwapCIAO ainda não estão integradas ao CIAO, e não há menção de outras formas de reconfiguração além da substituição de implementações de componentes previamente existentes, o que restringe as formas de reconfiguração. No trabalho descrito em [Hofmeister and Purtilo 1993] os pontos de reconfiguração são previamente codificados no programa, o que prejudica a flexibilidade da reconfiguração. Além disso, não é demonstrada aplicabilidade em sistemas de tempo-real. Em [Rasche e Polze 2005] utiliza-se .NET como plataforma para reconfiguração e avaliam o atendimento aos requisitos temporais de uma aplicação através do cálculo do seu tempo máximo de bloqueio.

Um aspecto importante da nossa abordagem é a separação dos mecanismos de reconfiguração e consistência, o que possibilita a utilização de diferentes estratégias para esses fins. Além disso, a comunicação com o ambiente de execução é mediada por um componente genérico, o que facilita sua integração em outras plataformas de componentes. Por fim, visto que o serviço de reconfiguração é baseado em componentes sua integração com outros serviços e, até mesmo, sua própria reconfiguração torna-se mais simples.

6. Considerações finais e perspectivas

Lidar adequadamente com as falhas de componentes de software em tempo de execução é de grande valia para sistemas com exigentes requisitos de disponibilidade, a exemplo de aplicações de controle em tempo real. Nesse sentido, serviços de reconfiguração dinâmica são ferramentas úteis na manutenção do desempenho da aplicação em níveis adequados ou, caso necessário, na sua adaptação de acordo com as novas condições do ambiente.

Este trabalho descreveu o projeto e a implementação de um serviço de reconfiguração no contexto de sistemas de tempo real distribuídos. Os mecanismos implementados foram avaliados no contexto de uma aplicação representativa do domínio: controle veicular. A análise dos resultados experimentais, através de métricas de qualidade de controle, revelou o atendimento adequado aos requisitos de desempenho da aplicação e o baixo impacto da implementação do serviço de reconfiguração.

Como trabalho futuro, avaliaremos o impacto dos mecanismos de reconfiguração dinâmica em outros cenários de reconfiguração.

Referências

Andrade, S. Macêdo, R. (2005) "A Component-Based Real-Time Architecture for Distributed Supervision and Control Applications". 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2005) p. 15-22.

Andrade, S. S., Macêdo, R. J., Sa, A. S. and Santos, N. P. (2006) "Using Real-time Components to Construct Supervision and Control Applications", 27th IEEE Real-Time Systems Symposium (RTSS 2006) Work in Progress Session.

Andrade, S., Macêdo R. (2007) "Engineering Components for Flexible and Interoperable Real-Time Distributed Supervision and Control Systems". In 12th IEEE Conference on Emerging Technologies and Factory Automation.

Andrade, S. S. (2006). “Sistemas Distribuídos de Supervisão e Controle Baseados em Componentes de Tempo Real”. Dissertação de Mestrado. Universidade Federal da Bahia.

Balasubramanian, J., Natarajan B., Parsons, J., Schmidt, D. C. and Gokhale A. (2005) "Middleware Support for Dynamic Component Updating", International Symposium on Distributed Objects and Applications (DOA).

Batista, T. V., Joolia A. and Coulson G. (2005) "Managing Dynamic Reconfiguration in Component-Based Systems", 2nd European Workshop on Software Architecture.

Batista, T., Cerqueira, R. and Rodriguez N. (2003) "Enabling Reflection and Reconfiguration in CORBA", In: 2nd Workshop on Reflective and Adaptive Middleware - ACM/IFIP/USENIX International Middleware Conference. pp 125 – 129.

Bidan, C., Issarny, V., Saridakis, T., Zarras, A. (1998) "A dynamic reconfiguration service for CORBA", IEEE International Conference on Configurable Distributed Systems.

Chan, A. S and Wu, G. 2002. "Architectural Level support for Dynamic Reconfiguration and Fault Tolerance in Component-Based Distributed Software". 9th International Conference on Parallel and Distributed Systems. IEEE Computer Society.

Coulson, G., Blair, G., Clarke, M. and Parlavantzas, N. (2002) "The design of a configurable and reconfigurable middleware platform". Distributed Computing Journal. Volume 15, Number 2, p. 109-126.

Hofmeister, C. and Purtilo, J. (1993) "Dynamic reconfiguration in distributed systems: Adapting software modules for replacement", 13th International Conference on Distributed Computing Systems, p. 101--110.

Hillman J. and Warren I. (2004) "An Open Framework for Dynamic Reconfiguration", In: 26th International Conference on Software Engineering (ICSE), pp. 594-603.

IIOP.NET Homepage. (2004) ".NET, CORBA and J2EE Interoperation". http://iiop-net.sourceforge.net.

Kramer, J. and Magee, J. (1990) "The Evolving Philosophers Problem: Dynamic Change Management", In: IEEE Transactions on Software Engineering. Volume 16, Issue 11.

Macêdo, R. J. A et al. (2004). "Tratando a Previsibilidade em Sistemas de Tempo Real Distribuídos: Especificação, Linguagens, Middleware e Mecanismos Básicos", In: 22º Simpósio Brasileiro de Redes de Computadores". Livro texto do minicurso, p. 105-163.

Object Management Group. (2006) "CORBA Component Model Specification", http://www.omg.org/cgi-bin/doc?formal/06-04-01.

Ogata, K. (2001). "Modern Control Engineering". 4th edition, Prentice-Hall, New Jersey.

Pont, M. G. (2001) "Patterns for time-triggered embedded systems", Addison-Wesley Professional, 1st edition.

Rasche, A. and Polze, A. (2005) "Dynamic Reconfiguration of Component-based Real-time Software", 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems.

Sans, Ricardo, Segarra, Miguel, Losert, Thomas, Bermejo, Julita, Arzén and Karl-Erik. (2003) "Engineering Handbook for CORBA-based Control Systems." Madrid:Universidad Politécnica de Madrid.

Schneider, E., Picioroaga, F., and Brinkschulte, U. (2004) "Dynamic reconfiguration through OSA+, a real-time middleware". 1st Int. Doctoral Symp. on Middleware.

Wang, N. (2004) "Composing Systemic Aspects Into Component-Oriented DOC Middleware", PhD thesis, St. Louis: Washington University.

Wang, N., Gill, C. (2004) "Improving Real-Time System Configuration via a QoS-aware CORBA Component Model", 37th Hawaii International Conference on System Sciences.

Wermelinger, M. A. (1999) "Specification of software architecture reconfiguration", Ph.D. thesis, Universidade Nova de Lisboa.