VI CONGRESSO NACIONAL DE ENGENHARIA MECÂNICA
VI NATIONAL CONGRESS OF MECHANICAL ENGINEERING
18 a 21 de agosto de 2010 – Campina Grande – Paraíba - Brasil
August 18 – 21, 2010 – Campina Grande – Paraíba – Brazil

REPLICAÇÃO RECONFIGURÁVEL EM SISTEMAS DE TEMPO REAL

Semíramis Ribeiro de Assis, sema@ufba.br[1]
Raimundo José de Araújo Macêdo, macedo@ufba.br[1]
Sérgio Gorender, gorender@ufba.br[1]

[1]LaSiD – Universidade Federal da Bahia (UFBA), Caixa Postal 15.064 – 91.501-970 – Salvador – BA – Brasil.

Resumo: *Falhas de computadores, sejam elas de software ou hardware, podem causar sérios prejuízos para o correto funcionamento de sistemas mecatrônicos, como, por exemplo, uma planta industrial. Replicação é uma técnica altamente utilizada para mascarar estas falhas, acarretando em sistemas mecatrônicos mais robustos. Arquiteturas de software baseada em componentes são uma poderosa técnica que permite a construção de software através do reuso e montagem de componentes existentes. Neste artigo, nós exploramos a possibilidade de distribuir componentes de software, como controladores, através de uma rede de computadores com o intuito de desenvolver e implementar sistemas mecatrônicos tolerantes a falhas. Nós desenvolvemos e implementamos um gerenciador de replicação para componentes (GesRep), capaz de prover replicação de componentes através de uma técnica conhecida como replicação passiva – onde uma réplica primária quando detectada faltosa é automaticamente substituída por uma backup sem intervenção de um operador humano. GesRep foi implementado e validado através do ARCOS (Arquitetura para Controle e Supervisão), uma plataforma baseada em componentes destinada para construção de aplicações de controle e supervisão. Dados coletados por experimentos com o GesRep mostram que as réplicas faltosas do ARCOS são automaticamente substituídas e a operação normal é retomada em menos de 1 segundo.*

Palavras-chave: *Tolerância a falhas, replicação de componentes, CORBA CCM.*

1. INTRODUÇÃO

A utilização cada dia maior de *softwares* pela indústria de automação vem mostrando uma nova necessidade: a incorporação de métodos que possam minimizar ou anular os efeitos de uma falha nos sistemas computacionais, evitando prejuízos financeiros e perca de produção. Estes métodos podem ser caracterizados com uma palavra chave na área de Sistemas Distribuídos: a redundância. Esta redundância remete ao uso de técnicas de replicação, que significa pluralizar partes de *hardware* ou de *software*, garantindo assim uma maior disponibilidade do objeto replicado.

Quando se fala em replicação para *softwares* ou suas partes, podem-se citar duas técnicas principais: replicação ativa e passiva. Na técnica de replicação ativa, apresentada por vários trabalhos (Schneider, 1993; Little, 1994), uma requisição enviada por um cliente é recebida e processada por todas as réplicas do grupo, sendo o resultado do processamento retornado por todas as réplicas ao cliente. Para que este processo seja feito da maneira correta, é preciso garantir que haja determinismo de réplicas, ou seja, que todas as réplicas irão receber a mesma seqüência de requisições, numa mesma ordem e obter um mesmo resultado de processamento dentro de um mesmo intervalo de tempo. O resultado, antes de ser repassado ao cliente, passa por uma votação para que o valor obtido esteja de comum acordo com a maioria das réplicas. Esta técnica suporta modelos de falhas bizantinas (arbitrárias), sendo necessárias $2f+1$ réplicas, e por parada (*crash*), necessitando apenas de $f+1$ réplicas, neste caso.

Schneider et al (1993) apresenta a técnica de replicação passiva, que se resume ao processo de recebimento e processamento das requisições do cliente por apenas uma réplica no grupo, denominada primária, responsável por retornar o resultado deste processamento ao cliente. As outras réplicas do grupo são denominadas *backup*, tendo o estado interno sincronizado com a primária, e a substituindo em caso de falhas, através de uma eleição para o novo líder. A técnica suporta apenas o modelo de falhas por parada (*crash*), que tem por premissa a necessidade de $f+1$ réplicas para que f falhas sejam suportadas.

Comparando as duas técnicas apresentadas, Schneider et al (1993) aponta a ativa como a que gera um menor *overhead* de recuperação em caso de falhas de réplicas, pois todas as réplicas recebem e processam as requisições, não sendo necessário nenhum procedimento de substituição de uma réplica faltosa.

Softwares construídos tendo como base uma arquitetura de *software* estão sujeitos a falhas, que podem ser de processos, objetos ou componentes, de acordo com o tipo de arquitetura utilizado. Replicar estes pontos passíveis de falhas é importante para manter uma maior disponibilidade da aplicação, fornecendo também um aumento da sua confiabilidade. Como exemplo, podemos citar a replicação dos componentes críticos de uma aplicação em diferentes

host, ou replicação dos processos essenciais para o correto funcionamento de um sistema mecatrônico. Marangozova (2003) apresenta o conceito de componente como um conjunto de objetos encapsulados capazes de prover um serviço bem definido. Possui interfaces claras que definem quais os serviços fornecidos e quais os requeridos, possibilitando seu acoplamento a outros componentes, com a finalidade de montar uma aplicação.

OMG (2007) diz que a vantagem deste tipo de arquitetura é a possibilidade de alteração do serviço provido por um dos componentes, sem que haja necessidade de intervenção na aplicação inteira, bastando que a interface disponibilizada não sofra modificação.

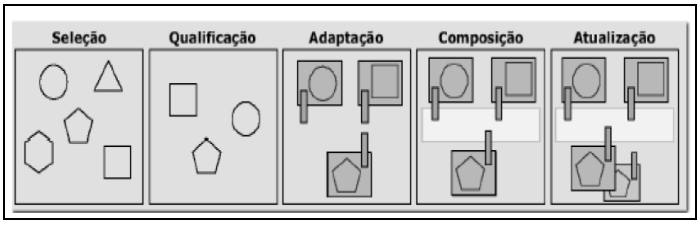

Figura 1 – Passos para construção de uma aplicação baseada em componentes, segundo OMG (2007).

A Fig. (1) ilustra os passos necessários para construção de uma aplicação baseada em componentes. O primeiro passo é a seleção dos componentes que podem fornecer os serviços necessários pela aplicação. Em seguida, devem-se qualificar estes componentes selecionados, verificando quais os que realmente atendem às expectativas da aplicação. Após isso, os componentes são adaptados para se adequarem às particularidades do *software*, compostos e, finalmente, podem ser atualizados para que novas versões de um componente possam substituir versões antigas.

Vários trabalhos (Orfali, 1998; OMG, 2007) apresentam a especificação *Commom Object Request Broker Architecture* (CORBA) como uma arquitetura de comunicação entre objetos distribuídos, de modo que estes objetos podem estar implementados em diferentes linguagens de programação e estarem em diferentes plataformas. OMG (2007) fala que a especificação CORBA para construção de aplicações baseadas em componentes é o *CORBA Component Model* (CCM), que define a criação de uma interface como base para o desenvolvimento de um componente. A partir desta interface é possível implementar um componente com o auxílio de ferramentas que dêem suporte ao CCM. Este componente criado irá funcionar implantado em um *container*, fornecido pelo CCM, responsável por gerenciar toda a parte de comunicação deste recém criado componente com os demais componentes a ele conectados, assim como com outros implantados neste *container*.

Uma das implementações disponíveis do CCM, conforme apresentado por vários trabalhos (Schmidt, 1998; Wang, 2003; Schmidt et al, 1998), é o *Component Integrated ACE ORB* (CIAO), que tem como propósito prover Qualidade de Serviço (QoS), através da extensão do ORB do *The ACE ORB* (TAO), uma implementação da especificação CORBA voltada para tempo real. O TAO, por sua vez, tem por base o *framework ADAPTIVE Communication Environment* (ACE), que é voltado para construção de aplicações baseadas em serviços de comunicação em tempo real e aplicações com requisitos temporais.

Tendo em vista o domínio de aplicações em tempo real *soft*, ou *soft real-time*, foi utilizado o conjunto ACE+TAO+CIAO para o desenvolvimento de um Gestor de Réplicas voltado para aplicações baseadas na arquitetura de componentes, projeto este denominado GesRep.

O projeto proposto tem por contribuição o fornecimento do serviço de replicação para componentes e suas conexões, tendo por base o modelo de componentes CCM, visando atender aplicações do domínio de controle e supervisão que possuam características de tempo real *soft*. Todos os trabalhos relacionados que foram estudados não estão voltados para replicação de componentes, mas sim para replicação de objetos simples (classes ou métodos), assim como *frameworks* voltados para construção de aplicações do domínio de controle e supervisão não forneciam este tipo de serviço para tolerância a falhas.

A validação do Gestor proposto foi realizada utilizado o *framework* para construção de aplicações voltadas ao domínio de controle e supervisão *Architecture For Control And Supervision* (ARCOS), apresentado por vários trabalhos (Andrade e Macêdo 2005; Andrade et al 2005), que tem sua arquitetura baseada em componentes e é voltado para construção de sistemas mecatrônicos com características de tempo real. Este *framework* possui uma aplicação teste construída, de nome ARCOS *Management Tool*, utilizada no processo de validação do GesRep.

O artigo está dividido da seguinte forma: na seção 2 serão apresentados trabalhos correlatos, a seção 3 traz o GesRep, suas funcionalidades, detalhes de implementação e usabilidade. A seção 4 apresenta os resultados obtidos com experimentos e, por fim, a seção 5 apresentará as conclusões obtidas.

2. TRABALHOS RELACIONADOS

Vários trabalhos (Cukier et al, 1998; Bagchi et al, 1998; Olsen et al, 1997; Little e MCCUE, 1994; Natarajan et al, 2000; Santos et al, 2005) apresentam diferentes mecanismos de gerenciamento de replicação, capazes de fornecer o serviço de replicação de objetos em um sistema. Estes mecanismos fazem parte de *frameworks* e arquiteturas criados para o desenvolvimento de sistemas distribuídos tolerantes a falhas, com diferentes requisitos e características.

O *Adaptive Quality of Service Availability* (AquA), apresentado por Cukier et al (1998), é uma arquitetura desenvolvida para prover o nível desejado de tolerância a falhas a aplicações distribuídas. A infraestrutura Proteus foi criada para prover tolerância a falhas ao AquA, apresentando um mecanismo de gerenciamento de replicação de objetos, o qual utiliza replicação ativa e passiva. As réplicas são organizadas em grupos, de modo que os grupos podem se comunicar entre si através de grupos de conexão, caso a comunicação seja necessária.

O Chameleon, apresentado por Bagchi et al (1998), é uma infra-estrutura de *software* adaptável, capaz de prover níveis diferentes de disponibilidade em um ambiente distribuído. Esta infra-estrutura é baseada no conceito de ARMORS - *Adaptive* (Adaptação), *Reconfigurable* (Reconfiguração) *and Mobile Objects for Reliability*. Componente chamados *Common ARMORS* são criados para prover serviços específicos aos sistemas, dando suporte à replicação de objetos, entre outros serviços.

Olsen et al (1997) apresenta o Piranha como um mecanismo para monitoração e recuperação de falhas, baseado no Corba. Este mecanismo atua com um gerenciador, reiniciando objetos CORBA que falharam, replicando objetos, movendo objetos entre hosts, entre outros serviços com o objetivo de prover a recuperação de falhas. A replicação dos objetos é dinâmica, formando grupos, provendo comunicação através de *multicast* confiável. Utiliza replicação passiva ou ativa.

Little e MCCUE (1994) desenvolveram o Sistema Gerenciador de Replicação (RMS), que tem por objetivo replicar objetos em uma rede, fornecendo QoS para cada um destes objetos. É possível configurar o número e a localização das réplicas. Este sistema utiliza as técnicas de replicação passiva e ativa, sendo a primeira para a replicação do próprio RMS.

Natarajan et al (2000) traz o *middleware* DOORS, que foi construído com base na especificação FT-CORBA, padronizada pela OMG (2007) com o objetivo de prover tolerância a falhas a sistemas CORBA. Este *middleware* provê a replicação de objetos CORBA, sendo composto por detectores de defeito e gerenciadores de replicação. Utiliza a técnica de replicação *warm_passive*, definindo uma réplica primária, como na replicação passiva e pontos de checagem (*checkpoints*). O usuário pode selecionar a técnica de monitoração a ser utilizada, podendo ser *heartbeat* ou *polling*.

FTWeb, como mostra Santos et al (2005), é uma infra-estrutura voltada para prover tolerância a falhas para aplicações baseadas em Web *Services*, através da replicação de objetos CORBA distribuídos em domínios diferentes. Utiliza a especificação FT-CORBA como base para sua implementação. Organiza as réplicas em grupos e possui alguns parâmetros configurados via arquivo XML, como a técnica de replicação, o estilo e intervalo de monitoramento, o tempo de resposta e indicação de recuperação. Implementa as técnicas de replicação *cold passive*, *hot passive* e ativa.

Estes mecanismos de replicação anteriormente apresentados, capazes de prover tolerância a falhas a sistemas distribuídos de uma maneira geral, criam e gerenciam réplicas de objetos destes sistemas. O gerenciador de réplicas que apresentamos neste artigo cria e gerencia réplicas de componentes de software e suas conexões, sendo por sua vez também desenvolvido segundo o conceito de componentes.

Os gestores estudados não são voltados para aplicações de tempo real, construídas sobre a arquitetura de componentes, não podendo ser utilizados para prover tolerância a falhas à aplicações do domínio de controle e supervisão que tenham sido desenvolvidas utilizando o *framework* ARCOS, motivo que induziu à construção do GesRep.

O processo de sincronização das réplicas, nos trabalhos relacionados estudados, acontece com suporte da infraestrutura utilizada, como o fornecimento do serviço de *multicast* confiável (*reliable multicast*), importante para correta sincronização entre réplicas, fornecido pelo ORB utilizado. O GesRep, entretanto, implementa toda a parte do gerenciamento e sincronização das réplicas, incluindo o protocolo de *reliable multicast*, garantindo que a mensagem de sincronização será recebida por todas as réplicas *backup*, retirando do ORB a necessidade de fornecimento deste protocolo.

3. GESTOR DE RÉPLICAS - GESREP

Uma aplicação baseada em componentes pode ter todos ou alguns componentes instalados de forma distribuída, de modo que uma falha em um destes componentes distribuídos pode acarretar a parada completa da aplicação. Para evitar que este problema ocorra, com um possível prejuízo para a indústria, como parada da planta de produção, a replicação destes componentes pode ser realizada através de um Gestor de Réplicas.

O GesRep, construído com base na arquitetura de componentes e no conjunto ACE+TAO+CIAO, tem por finalidade criar, destruir, sincronizar e monitorar réplicas de componentes, que tenham por base a mesma arquitetura do Gestor. As réplicas criadas se tornam membros de grupos homogêneos (contendo apenas um tipo de componente), ficando a cargo do Gestor o gerenciamento destes grupos e seus membros e a consistência entre as réplicas. Um grupo de réplicas pode ter seu número mínimo de réplicas reconfigurado dinamicamente, de acordo com informação do usuário, de modo a atender um crescimento ou diminuição de maquinas na rede, contanto que estas estejam previamente cadastradas em arquivo de configuração. Por seguir a especificação da política de replicação passiva, e o modelo de falhas por parada, o Gestor restringe em, no mínimo, duas réplicas em cada grupo, fazendo jus à premissa de $f+1$ réplicas para tolerar f falhas.

As características de reusabilidade e de extensão estão presentes no GesRep, já que, por ter sua arquitetura baseada em componentes, é possível que novas funcionalidades sejam acopladas, através da adição de novos componentes e sem necessidade de alteração de toda a aplicação, assim como os componentes atualmente presentes podem ser reutilizados em outras aplicações baseadas em componentes.

Através de um arquivo de configuração em XML, é possível informar ao GesRep os dados do(s) componente(s) a ser (em) replicado(s), a técnica de replicação escolhida, além de informar se há uma reconfiguração ou não na quantidade de réplicas de um grupo. Este arquivo passa a ser monitorado em intervalos periódicos para detectar alguma possível reconfiguração e, neste caso, tomar as medidas necessárias para adequação do grupo alterado, aumentando ou diminuindo o número de membros no mesmo. Embora a versão atual do Gestor implemente a técnica de replicação passiva, sua construção foi feita com base na possibilidade de extensão de suas funcionalidades, de modo que novas técnicas de replicação possam ser adicionadas e, então, o usuário possa selecionar a melhor para sua aplicação.

A criação e remoção de réplicas são feitas com o auxílio de um serviço do CIAO, de nome *Redeployment and Reconfiguration* (ReDaC). Este serviço facilita a alteração dinâmica do plano de execução da aplicação, tarefa custosa e propensa a erros caso feita de forma manual, permitindo adicionar e remover instâncias de componentes, assim como suas conexões com os demais componentes da aplicação, como afirma Wang et al (2003).

A Fig. (2) apresenta a política de consistência entre as réplicas, que é mantida da seguinte forma: após a réplica primária processar uma requisição em que o estado interno tenha sido alterado (1,2), anteriormente ao envio do resultado ao cliente, este é enviado através de mensagens de atualização às réplicas *backup* (3), sendo armazenado em seguida em meio persistente (4). O resultado é retornado ao cliente (5). O componente Grupo é responsável pela recuperação do estado armazenado (6) e ressincronização periódica das réplicas *backup* (7) e de uma nova réplica que tenha sido incorporada ao sistema (8). Em caso de incorporação de nova réplica ao sistema, esta receberá mensagem de sincronização de estado no próximo processamento de requisição com alteração de estado interno ocorrido na réplica primária (9). O estado interno de uma réplica representa o valor das variáveis de escopo global da(s) classe(s) executora(s) do componente.

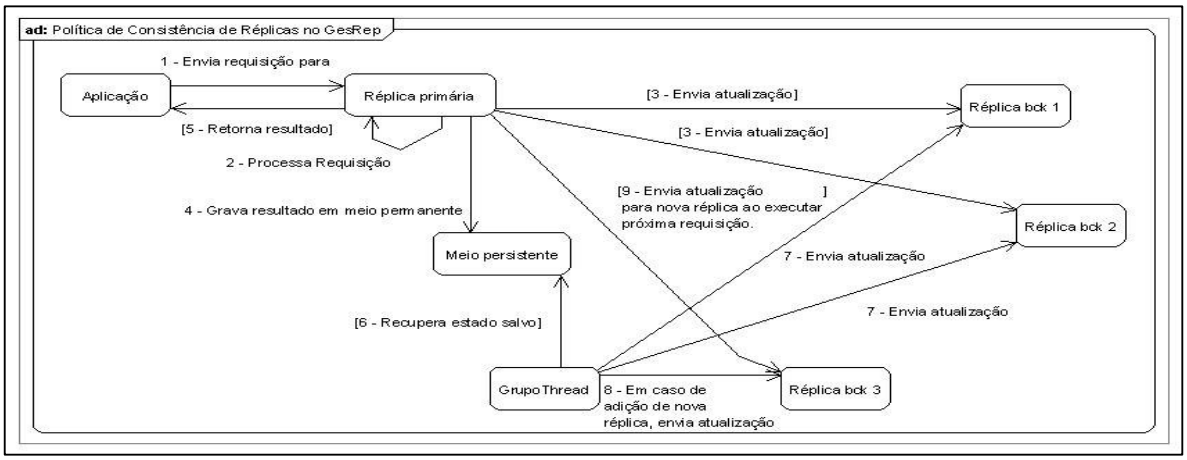

Figura 2 – Política de replicação implementada no GesRep.

3.1. Arquitetura

O GesRep está dividido em dois módulos: Administrador e GestorQos. O primeiro módulo abrange funcionalidades básicas para um Gestor de Réplicas, como a criação e remoção de réplicas e grupos, adição e remoção de membros em grupos, atualização de conexões entre réplicas e sincronização de estado interno entre membros *backup* com o primário. Este módulo é formado pelos componentes Réplica e Grupo.

O segundo módulo abrange a reconfiguração de grupos, através da alteração da quantidade mínima de réplicas em um grupo, e o serviço de monitoramento das réplicas. Cada réplica criada pelo GesRep possui um componente monitor associado à mesma, ficando este responsável por checar a ocorrência de falhas por parada (*crash*) na máquina hospedeira ou no *container* no qual a réplica está instalado e, em caso positivo, tomar as providências necessárias para a recuperação da quantidade mínima de membros do grupo. É composto pelos componentes GestorQoS e Monitor.

Além destes dois módulos, também faz parte da arquitetura do Gestor a interface Replicação, que tem a função de facilitar a comunicação do Gestor com o componente que está sendo replicado, através da implementação de alguns métodos. Dentre estes, estão o método de *ping*, métodos para salvar e recuperar o estado interno e para configurar a técnica de replicação utilizada para o componente. Esta interface deve ser estendida e suportada pelo componente que desejar utilizar o serviço de replicação provido pelo GesRep, e seus métodos devem ser implementados pela classe executora deste componente.

A Fig. (3) ilustra a arquitetura anteriormente descrita do Gestor. No módulo Administrador, os componentes Réplica e Grupo estão interconectados. O componente Grupo cria um grupo de réplicas e solicita ao componente Réplica que crie as réplicas do componente solicitado, juntamente com as conexões necessárias. Após isso, este componente é inserido no grupo previamente criado, que solicita ao componente Monitor a criação de um monitor para a réplica recém criada. O componente Monitor responsável por criar os monitores para as réplicas solicita ao

GestorQoS, por sua vez, que verifique uma possível alteração na quantidade mínima de réplicas em um grupo, caracterizando uma reconfiguração do grupo.

A interface Replicação é utilizada como uma configuração do componente que irá ser replicado, servindo como um facilitador na comunicação GesRep x Réplica. Por fim, este componente estará interligado ao restante da aplicação, caracterizando o papel do cliente.

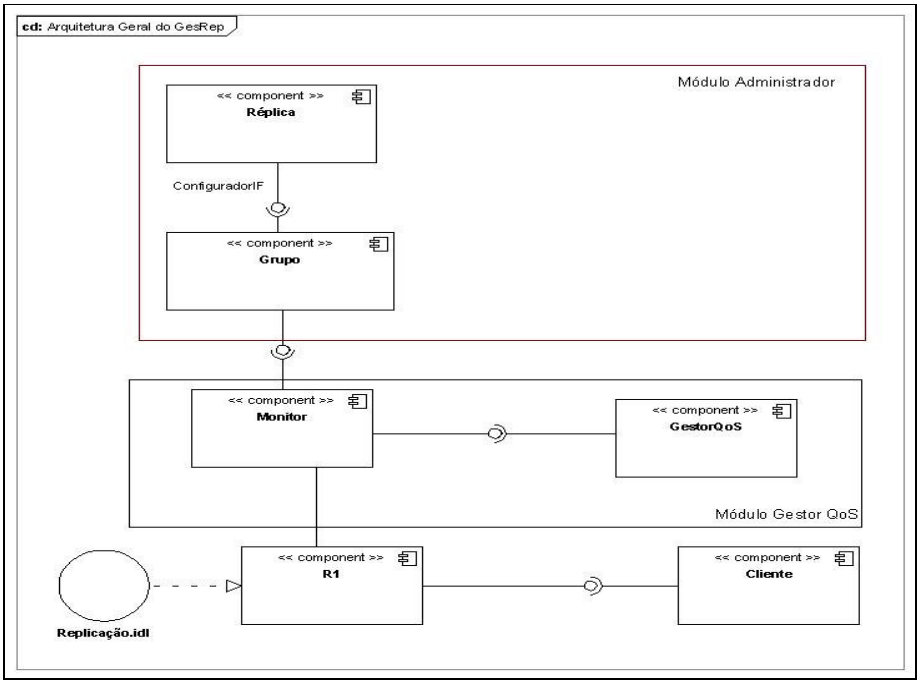

Figura 3 – Arquitetura do GesRep.

3.2. Implementação e Usabilidade

O Gestor foi implementado de forma independente de aplicação, de modo a permitir que qualquer componente de qualquer aplicação construída baseada na arquitetura ACE+TAO+CIAO possa ser replicado.

A configuração do GesRep é feita através de arquivo XML, contendo os dados do componente que será replicado. Estes dados são obtidos através do arquivo descritor de implantação da aplicação (extensão .CDR) que terá o componente replicado.

A configuração do componente a ser replicado, é feita através dos arquivos executor, MPC e descritor de implantação da aplicação. No arquivo executor, é necessário implementar os métodos herdados da interface Replicação. No arquivo MPC, que é gerado no momento da criação do componente, é preciso que a biblioteca do Gestor seja incluída na lista de bibliotecas utilizadas pelo componente para sua compilação.

```
<REPLICA>
        <UUID></UUID>
        <INSTANCIA></INSTANCIA>
        <IMPLEMENTATION></IMPLEMENTATION>
        <QTDMINREPLICAS></QTDMINREPLICAS>
        <NODENAME></NODENAME>
        <RECONFIGURACAO></RECONFIGURACAO>
        <TIPO_REPLICACAO></TIPO_REPLICACAO>
</REPLICA>
```

Figura 4 – Arquivo XML de configuração do GesRep.

A Fig. (4) ilustra a estrutura do arquivo XML de configuração do GesRep, que deve ser preenchido com as informações sobre o componente a ser replicado. Estas informações são obtidas através do arquivo descritor de implantação da aplicação à qual pertence o componente.

A Fig. (5) ilustra um exemplo de um arquivo descritor de implantação, de onde as informações para o preenchimento do arquivo XML do GesRep são retiradas. O campo ***name*** contém a informação que será utilizada no campo INSTANCIA da Fig. (4). O campo ***implementation*** contém a informação que será colocada no campo equivalente na Fig. (4). O campo UUID da Fig. (4) será preenchido com o identificador único do arquivo descritor, sendo este localizado no início do arquivo. O campo QTDMINREPLICAS representa a quantidade mínima de réplicas no grupo. O campo NODENAME indica em qual *host* o componente replicado será instalado, sendo este *host* previamente cadastrado no sistema. O campo RECONFIGURACAO recebe os valores 1, para o caso de mudança na

quantidade mínima de réplicas no grupo e 0 para uma situação sem alteração. O TIPO_REPLICACAO indica qual será a técnica de replicação utilizada, também recebendo um número inteiro. A técnica de replicação passiva, que está implementada no Gestor, equivale ao valor 0, porém outras técnicas de replicação podem ser implementadas e outros valores numéricos a elas atribuídos.

```
281   <instance id="ARCOS-DAISSimulatedCarProvider-idd">
282     <name>ARCOS-DAISSimulatedCarProvider-idd</name>
283     <node>MainNode</node>
284     <source><!-- @@ What goes here --></source>
285     <implementation>ARCOS-DAISSimulatedCarProvider-mdd</implementation>
```

Figura 5 – Parte de um arquivo descritor de implantação.

Após a configuração do arquivo XML, é necessário que o componente implemente os métodos da interface Replicação, do GesRep. Esta ação irá permitir ao Gestor o acesso a alguns métodos fundamentais para o processo de gerenciamento da réplica, como o processo de sincronização de estado interno entre uma nova réplica criada e uma já existente no grupo, além de realização de *ping* no componente replicado.

```
interface ReplicacaoIF
{
  string salvaEstado();
  long recuperaEstado(in string estadoSalvo);
  long estaVivo();
  void setTipoReplicacao(in long tipoReplicacao);
  long getTipoReplicacao();

};
```

Figura 6 – Interface Replicação.

A Fig. (6) ilustra a interface Replicação. Os métodos de salvar estado e recuperar estado têm por finalidade armazenar o estado interno do componente, permitindo que este estado seja sincronizado com uma nova réplica criada.

O estado interno de uma réplica representa suas variáveis globais, tanto a nível de classe como de *namespace*. Os valores das variáveis devem ser agrupados em uma *string* e concatenados com o caractere *pipe* ('|'). A *string* resultante desta concatenação deve ser armazenada em meio persistente. Esta mesma *string* será recebida pelo método de recuperação de estado, e os valores das variáveis armazenadas serão recuperados na mesma ordem na qual foram concatenados anteriormente.

Uma *thread* responsável pela sincronização irá, então, recuperar o dado armazenado e repassar os valores a todas as réplicas *backup* do grupo, de forma periódica, além da troca de mensagens entre a réplica primária e as *backups* a cada invocação de método que altere o estado interno da réplica principal.

O método *estaVivo* tem por objetivo permitir ao GesRep verificar se houve falha por parada no *host* que hospeda o componente monitorado, ou no *NodeManager* (*container*) no qual a réplica está instalada, representando um *ping*.

Os métodos *setTipoReplicacao* e *getTipoReplicacao* têm a finalidade de informar ao componente qual a técnica de replicação que está sendo utilizada pelo Gestor para o componente específico.

Após a modificação do componente com a implementação dos métodos acima descritos, é preciso alterar o arquivo de extensão MPC do componente, inserindo a biblioteca do GesRep que representa a interface Replicação na lista de bibliotecas para compilação do componente.

3.3. Funcionamento

Após as configurações do GesRep e do componente cliente, o funcionamento do Gestor se dará da seguinte forma: o arquivo gesRep.XML será lido por uma aplicação presente no componente Réplica do módulo Administrador. Após este procedimento, os grupos e suas respectivas réplicas serão criados, a instância do componente a ser replicado que foi inicialmente criado no momento da implantação da aplicação será removida e suas conexões migradas para a réplica primária criada pelo GesRep, passando esta a estar conectada diretamente aos demais componentes da aplicação. Este procedimento de remoção é necessário devido à instância inicial não estar no padrão de nomenclatura controlado pelo Gestor.

Para cada réplica criada, um novo componente do tipo Monitor será criado e associado a esta, ficando este responsável pela detecção de falha da máquina que hospeda a réplica monitorada ou do *container* no qual está instalado o componente. Os componentes do tipo Monitor serão sempre criados no *MainNode*, nodo principal da rede, no qual estão inicializados os servidores das aplicações cliente e GesRep e *ExecutionManager* do CIAO. Em caso de falha detectada, duas ações poderão ser tomadas, a depender do tipo da réplica faltosa:

- **Réplica primária** – As conexões feitas entre a réplica falha e as demais réplicas da aplicação serão migradas para a nova líder do grupo, enquanto que um novo membro *backup* será criado para manter a quantidade mínima de réplicas no grupo. A eleição da nova líder é feita de forma automatizada, sendo ela o membro

referenciado pela segunda posição no grupo. Após o processo de recuperação dos membros do grupo, o componente monitor responsável pela réplica falha, requisita sua remoção do sistema, seguindo-se da remoção da *thread* ligada a este monitor.

- **Réplica *backup*** – Um novo membro é criado e inserido no grupo, como forma de manter o número mínimo de réplicas no grupo. O componente monitor responsável, assim como a sua respectiva *thread*, se autodestroem após a recuperação do número de membros do grupo.

Além do monitoramento das réplicas, há o monitoramento do arquivo de configuração do GesRep, objetivando detectar uma reconfiguração do número mínimo de réplicas em um determinado grupo, informada pelo usuário da aplicação.

O conjunto ACE+TAO+CIAO não foi desenvolvido visando prover de recursos de tolerância a falhas. Com isso, não é possível replicar processos essenciais para a implantação de uma aplicação numa mesma rede, como, por exemplo, o *ExecutionManager*, responsável por gerenciar o plano de execução da aplicação e os *containers* distribuídos na rede. Com isso, para evitar que os processos que dão suporte ao GesRep se tornem ponto único de falhas, por estarem concentrados em uma única máquina, é necessário que toda a estrutura do Gestor e da aplicação que dele utiliza o serviço de replicação seja replicada em uma rede diferente. Assim, em caso de falhas na máquina principal de uma das redes, a outra poderá tolerar através do bom funcionamento. Não é objetivo do grupo mantenedor do conjunto ACE+TAO+CIAO a inclusão de mecanismos de tolerância a falhas, nem há previsão para que isso aconteça.

4. RESULTADOS

Os testes no GesRep, como citado anteriormente, foram realizados utilizando o *framework* ARCOS, apresentado por vários trabalhos (Andrade e Macêdo 2005; Andrade et al 2005), através da replicação de um dos componentes essenciais para o funcionamento da aplicação teste desenvolvida utilizando este *framework*, de nome ARCOS *Management Tool*. Através da replicação de um componente do ARCOS, busca-se uma maior disponibilidade da aplicação anteriormente citada, considerando que os componentes desta estão instalados em diferentes máquinas, e a falha de uma destas máquinas implica em parada completa desta aplicação do ARCOS. A replicação, caso presente, auxilia na disponibilidade do componente replicado, recriando o mesmo em uma máquina diferente da faltosa.

O componente selecionado para validação foi o ARCOS DAIS *Simulated Car Provider*, que tem a finalidade de prover valores simulados para as variáveis velocidade *(speed*) e acelerador (*throttle*) de um carro também simulado.

Para que os experimentos fossem realizados, foi necessário que todas as máquinas presentes na rede estivessem cadastradas previamente no arquivo de configuração NodeManagerMap.dat, localizado na pasta *descriptors* do ARCOS, local onde o serviço *ExecutionManager* do CIAO é invocado.

Os experimentos foram realizados com 5 máquinas, variando entre 2 e 10 réplicas do componente citado, e tiveram por objetivo avaliar os tempos de resposta do GesRep para as operações de criação e sincronização de grupo de réplicas, além da sincronização individual e recuperação de uma réplica, em caso de falhas. Outro objetivo importante é a continuidade do funcionamento da aplicação em caso de falhas de uma réplica, sendo ela primária ou *backup*. Para cada um dos experimentos, foram coletados resultados de 60 amostras consecutivas. A rede na qual estas máquinas se encontram estava isolada, sem acesso à internet ou outras máquinas, com velocidade de 100 Mbps e foi assumida a premissa de não ocorrência de falhas na comunicação, consequentemente, sem perdas de mensagens.

Para todos os resultados obtidos, os tempos foram mensurados em microssegundos (µs), sendo convertidos para segundos com uma precisão de cinco casas decimais, para uma melhor visualização.

Tabela 1 – Tempo de criação de um grupo de réplicas.

Tempo de Criação - Grupo de Réplicas				
Num. Máquinas	**Num. Réplicas**	**Média**	**Desvio padrão**	**Mediana**
2	2	0,27842	0,04284	0,28693
3	3	0,68027	0,07426	0,7016
4	4	1,32662	0,04439	1,33359
5	5	1,77005	0,07785	1,75854
5	6	1,88385	0,05355	1,87942
5	7	2,07954	0,06435	2,08425
5	8	2,28458	0,06471	2,27395
5	9	2,50607	0,05988	2,51085
5	10	2,83483	0,06916	2,83804

A Tab. (1) apresenta os tempos, em segundos, obtidos com a criação de um grupo completo de réplicas. Como o desvio padrão tem um valor bastante inferior ao da média, assumimos que os vários experimentos tiveram um comportamento constante. Observamos também que o tempo para criação de um grupo de réplicas cresce com o aumento no número de réplicas no grupo. Entretanto, este crescimento só é superior a 0,4 segundos com o aumento de 3 para 4 máquinas e réplicas. Nas demais variações, este valor varia entre 0,1 e 0,4 segundos.

Tabela 2 – Tempo de sincronização de um grupo de réplicas.

Sincronização de Grupo			
Qtd. Réplicas	**Média**	**Desvio Padrão**	**Mediana**
2	0,00543	0,00133	0,00586
3	0,00883	0,00009	0,00884
4	0,01229	0,00035	0,01234
5	0,01958	0,00156	0,01868
6	0,01401	0,00104	0,01415
7	0,02615	0,00229	0,02694
8	0,02092	0,00608	0,01815
9	0,03239	0,00173	0,03216
10	0,03926	0,00279	0,03871

A Tab. (2) apresenta os resultados de tempo obtidos com a sincronização de um grupo de réplicas. Novamente observamos que o desvio padrão apresenta um valor consideravelmente inferior ao da média, caracterizando experimentos consistentes, com valores constantes. O valor da média também cresce com o número de réplicas, sendo que, de 5 para 6 réplicas, e de 7 para 8 réplicas, houve uma redução no tempo médio de sincronização.

Tabela 3 – Tempo de recuperação de uma réplica em caso de falhas.

Tempo de Recuperação de Réplica			
Tipo	**Média**	**Desvio Padrão**	**Mediana**
Backup	0,17876	0,03946	0,20046
Primária	0,16078	0,03937	0,156

Nesta Tab. (3) estão apresentados os cálculos do tempo de recuperação de uma réplica do tipo primária e do tipo *backup*. Na recuperação de uma primária, é necessário que haja a migração das conexões que envolviam a réplica faltosa para a primeira réplica *backup* do grupo, que passará a ser a nova primária, e a criação de uma nova réplica *backup*, garantindo a manutenção do número mínimo de réplicas no grupo. A operação de recuperação em caso de falha de uma réplica *backup* equivale à criação de uma nova réplica *backup* para manutenção do número mínimo de réplicas no grupo.

Tabela 4 – Tempo de recuperação de réplicas na presença de alta carga de processamento e alto tráfego na rede.

Tempo de Recuperação de Réplicas na Presença de Stress de Processamento e alto tráfego na Rede.			
Tipo	**Média**	**Desvio Padrão**	**Mediana**
Backup	0,2372	0,03055	0,23739
Primária	0,24003	0,02601	0,23752

O resultado apresentado na Tab. (4) indica o tempo de recuperação para réplicas do tipo *backup* e primária, considerando uma alta carga de processamento nas máquinas e alto tráfego de mensagens na rede.

Se comparadas as tabelas 3 e 4, verifica-se que, mesmo na presença de carga significativa de processamento pela máquina e de alto tráfego na rede, a performance do GesRep não é significativamente alterada, mantendo um comportamento adequado para o bom funcionamento da aplicação que dele faça uso.

Verificou-se a continuidade do funcionamento da aplicação utilizada para teste, mesmo com falha da réplica primária. Esta disponibilidade se dá devido ao tempo mínimo de recuperação de falha na réplica primária, apresentado nas tabelas 3 e 4.

Os tempos de criação de um grupo completo de réplicas indicam que o processo de criação do grupo e inicialização das atividades do Gestor é feito de forma rápida, apesar de uma boa fatia deste tempo ser tomada pela execução do ReDaC.

Os tempos de sincronização de um grupo de réplicas mostram que, mesmo para um grupo com uma maior quantidade de réplicas, estas são sincronizadas quase que simultaneamente, levando apenas fração de segundo para que todo o processo seja completo.

5. CONCLUSÕES

Neste trabalho foi desenvolvido um Gestor de Réplicas (GesRep), voltado para atender às aplicações do domínio de Controle e Supervisão Industriais, construído baseado na arquitetura de componentes para ser utilizado pela plataforma ARCOS (*Architecture for Control and Supervision*). Com o GesRep se podem implementar mecanismos de tolerância a falhas para aplicações mecatrônicas, aumentando a disponibilidade de tais aplicações.

Através dos experimentos realizados utilizando o GesRep e a plataforma ARCOS foi possível concluir que os tempos de resposta do Gestor foram satisfatórios, não apresentando atrasos relevantes na execução das aplicações do ARCOS.

É possível ainda incrementar as funcionalidades do GesRep com novas políticas de replicação, como replicação ativa, que deixamos para trabalhos futuros.

6. REFERÊNCIAS

Andrade, S. S. et al, 2005. "ARCOS : A Component-based Architecture for the Construction of Robust Control and Supervision Applications.", Workshop on Dependable Automation Systems, 2nd Latin-American Symposium on Dependable Computing, Outubro.

Andrade, S. S. , de Araújo Macêdo, R. J., 2005 . "A Component-Based Real-Time Architecture For Distributed Supervision And Control Applications", 10th IEEE Conference on Emerging Technologies and Factory Automation.

Bagchi, S. Et al, 1998, "The Chameleon Infrastructure for Adaptive, Software Implemented Fault Tolerance", Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, pp. 261, Outubro.

Cukier, Michael Et al, 1998, "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects", Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, pp. 245, Outubro.

Little, M. C.; MCCUE, D. L., 1994. "The Replica Management System: a Scheme for Flexible and Dynamic Replication", Proceedings of the Second Workshop on Configurable Distributed Systems, Março.

Marangozova, Vânia, 2003, "Component Quality Configuration: the Case of Replication", Proceedings of WCOP 2003 Eight International Workshop on Component-Oriented Programming, Julho.

Natarajan, Balachandran et al, 2000. "DOORS: Towards High-performance Fault Tolerant CORBA". International Symposium on Distributed Objects and Applications, Setembro.

Object Management Group (OMG), 2007, "CORBA Overview", http://cgi.omg.org/docs/formal/99-07-06.pdf**,** Dezembro.

Olsen, Silvano Maffeis. Et al, 1997, "Piranha: A CORBA Tool for High Availability", Proceedings of IEEE Computer, Vol. 30, N• 4, Abril.

Orfali, Robert; HARKEY, Dan, 1998, "Client/ Server Programming with JAVA and CORBA", 2ª edição. Estados Unidos, Wiley.

Santos, Giuliana Teixeira et al, 2005. "FTWeb: A Fault Tolerant Infrastructure for Web Services". Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05).

Schmidt, Douglas, 1998, "An Architectural Overview of the ACE framework – A case-study of Successful cross-platform Systems Software Reuse", USENIX login magazine, Tools special issue, Novembro.

Schmidt, Douglas C. et al 1998, "The Design of the TAO Real-Time Object Request Broker", Computer Communications, Elsevier Science, Abril.

Schneider, Fred B. Et al, 1993. "The Primary - Backup Approach", Distributed Systems S. Mullender ed. Addison Wesley, pp. 199-216.

Schneider, Fred B., 1993, "Replication Management Using the State-Machine Approach", Distributed Systems, Addison-Wesley, 2nd Edition, pp. 169-197.

Wang, Nanbor; at. Al, 2003, "Total Quality of Service Provisioning in Middleware and Applications", The Journal of Microprocessors and Microsystems, pp. 45-54, Março.

7. DIREITOS AUTORAIS

RECONFIGURABLE REPLICATION IN REAL TIME SYSTEMS

Semíramis Ribeiro de Assis, sema@ufba.br[1]
Raimundo José de Araújo Macêdo, macedo@ufba.br[1]
Sérgio Gorender, gorender@ufba.br[1]

[1]LaSiD – Universidade Federal da Bahia (UFBA), Mailbox 15.064 – 91.501-970 – Salvador – BA – Brasil.

Abstract: *Computer hardware or software failures can cause serious damage for the correct functioning of mechatronic systems such as an industrial plant. Replication is a technique that has been widely used to mask such failures, leading to more robust mechatronic systems. Software component architectures are a powerful technique thet alows the construction of software from the reuse and assembly of existing components. In this paper, we explore the possibility of distributing software components such as controllers over a computer network in order to design and implement fault-tolerant mechatronic systems. We designed and implemented a component replication manager (GesRep) that provides component replication through a technique known as the passive replication - where a primary replica when detected faulty is automatically replaced by a backup one without the intervention of a human operator. GesRep has been implemented and evaluated over ARCOS (Architecture For Control And Supervision), a component-based platform devoted for the construction of distributed control and supervision applications. Data collected from experiments with GesRep show that ARCOS faulty replicas are automatically replaced and normal operation resumes in less than 1 second.*

Keywords: *Fault tolerance,components replication, CORBA CCM.*