UMA ARQUITETURA FLEXIVEL PARA SISTEMAS DISTRIBUIDOS TOLERANTES A FALHAS

Autor: Orlando G. Loques F. *

Palavras Chaves: Especificação e Projeto de Sistemas Tolerantes a Falhas, Software Tolerante a Falhas, Arquitetura de Computadores, Sistemas Distribuídos.

Resumo: Este trabalho apresenta a arquitetura de um sistema distribuído tolerante a falhas, o qual suporta redundância dos tipos "hot e cold standby" para tarefas definidas em software. Os dois tipos de redundância são suportados por um conjunto comum de mecanismos os quais tratam da deteção de falhas e da reconfiguração das tarefas da aplicação. Tarefas do tipo cold standby podem ser criadas e ativadas automáticamente pelo sistema de maneira a substituir tarefas em falha, porém informação de estado não é preservada. Para tarefas do tipo hot standby a informação de estado é preservada e uma técnica especial permite que a recuperação de falhas seja obtida de modo transparente. A principal atração desta técnica é permitir que as tarefas do sistema distribuído sejam programadas sem se ter tolerância a falhas em mente; em um estágio posterior do desenvolvimento do sistema as tarefas podem ser transformadas automáticamente de modo a se obter a capacidade de tolerância a falhas.

Abstract: This paper presents the software architecture of a fault-tolerant distributed system providing both cold and hot standby redundancy of software tasks. The two types of redundancy are supported by common mechanisms, which provide for detection of failures and for reconfiguration of the software tasks of the application. Cold standby tasks are created and activated by the system in order to replace failed tasks, but no state information is preserved. Hot standby tasks do preserve state information and provide transparent failure recovery. Transparent recovery is achieved through the use of a special technique that allows the program tasks to be programmed without fault-tolerance in mind; afterwards they can be automatically transformed in order to achieve that capability.

* Engenheiro Eletrônico (PUC/RJ, 1973), Mestre em Engenharia Elétrica (PUC/RJ, 1976), Ph.D. (Imperial College, 1984); arquitetura de sistemas distribuídos, sistemas operacionais, computação tolerante a falhas; Professor do Depto. de Eng. Elétrica da PUC/RJ, Rua Marquês de São Vicente, 225 - CEP 22453 - Rio de Janeiro, Rj.

1. Introdução

Sistemas distribuídos de computadores oferecem varias vantagems em relação a sistemas centralizados. O paralelismo inerente permite melhor desempenho, a capacidade local de processamento permite respostas mais rápidas no tempo, e a modularidade do software e hardware permite que o sistema seja facilmente expandido e modificado a um custo baixo. Uma das maiores vantagens da distribuição e do acoplamento fraco é o potencial para a obtenção da capacidade de tolerância a falhas. Devido ao acoplamento fraco, a falha de componentes não críticos pode levar a operação em modo degradado, ao invés da falha total do sistema. Por outro lado, os componentes críticos podem ser feitos tolerantes a falhas pelo uso de redundância no hardware, a qual devido ao baixo custo dos componentes pode ser suportada nas partes do sistema onde for requerida.

Este trabalho apresenta uma classificação de sistemas em relação aos seus requisitos de confiabilidade. Com base nesta classificação, uma arquitetura "software" para um sistema distribuído de computadores é apresentada. Essa arquitetura é baseada em Conic [6], um ambiente para a programação e configuração de sistemas especialmente projetado para o suporte de sistemas distribuídos embutidos. Contudo, os mesmos princípios são aplicáveis á construção de sistemas tolerantes a falhas baseados em outros ambientes. Os detalhes relevantes de Conic são descritos em um apêndice do trabalho.

2. Tipos de Sistema

A classificação captura a relação entre a falha de qualquer dos módulos que compoêm um sistema e a falha da especificação de confiabilidade associada ao mesmo sistema como um todo. Dois tipos de sistema são identificados:

Baixa Dependência a Falha: O sistema é tal que o padrão de confiabilidade exigido pela aplicação que ele implementa não é violado se um ou mais de seus módulos falha. Isto é, a capacidade de serviço oferecida pelo sistema é de alguma forma degradada devido a falha de um módulo. Contudo, ela ainda é suficiente considerando os requisitos de confiabilidade da aplicação.

Alta Dependência a Falha: O sistema é tal que o padrão de confiabilidade exigido pela aplicacão que ele implementa é violado se qualquer um dos módulos que o constitue falha. Isto é, a falha de um módulo leva a falha do sistema.

Em geral, a classificação apresentada acima simplifica a discussão e a apresentação de sistemas tolerantes a falhas. Embora a maioria dos sistemas possa ser enquadrada no tipo BDF, arquiteturas para o suporte de sistemas do tipo ADF também são necessarias [3,4,11], e alguns aspectos da construcão deste tipo de sistema são pesquisados, eg. [10]. Contudo, deve ser mencionado que sistemas de ambos os tipos podem conviver em uma mesma aplicacão. A arquitetura aqui apresentada pode suportar os dois tipos de sistema.

3. Hipoteses

Para o projeto do sistema foram adotadas as seguintes hipóteses: (i) O projeto é correto, i.e., erros de algoritmo [1] não são tratados, (ii) Confinamento de erros, é assumido que a comunicação entre as tarefas é feita exclusivamente através de mensagens, e que a falha de uma tarefa não causa a falha de outra tarefa através da propagação de erros, i.e, erros não se propagam através de mensagens. Diversas técnicas para a deteçaõ de erros são disponiveis, eg., [2,9]. (iii) Sistema de comunicação confiavel, i.e., ele deve deve ter caracteristicas que não comprometam a confiabilidade do sistema de aplicação: Em particular, é requerido que mensagens em transito não sejam corrompidas e que a rede de comunicação não se particione. O projeto deste tipo de sistema de comunicação é objeto de diversas pesquisas, eg., [5,8].

A hipótese (i) garante que se outros erros não ocorrerem as mensagens serão corretas. Considerando a hipótese (ii) a primeira consequência da falha de uma tarefa será a interrupção do serviço fornecido pelo mesma (falhas transientes podem ser tratadas internamente á estação). Então, considerando (ii) e (iii) falhas de tarefas podem ser detetadas sem ambiguidade pela verificação de parametros de tempo medidos localmente em outras tarefas. Em adição, uma técnica simples (checkpointing) pode ser usada para a recuperação de falhas em sistemas do tipo ADF.

4. O Sistema Tolerante a Falhas

De maneira a fornecer a capacidade de tolerância a falhas foram adicionados ao sistema Conic original dois tipos especiais de tarefas. Ao nivel lógico elas são identicas as tarefas originais de Conic. Contudo, cada tipo especial de tarefa possui uma implentação que lhe adiciona uma forma particular de redundância e supre requisitos diferentes de confiabilidade:

Cold Standby: O qual visa o suporte de sistemas tipo BDF.
Hot Standby: O qual visa o suporte de sistemas tipo ADF.

O projetista especifica cada tipo de tarefa a ser usada na configuração do sistema da aplicação de modo a suprir os requisitos da mesma. Ferramentas padronizadas de desenvolvimento são então usadas para realizar as transformaçoẽs necessárias para a implementação de cada tipo. Mecanismos para suporte durante a operação do sistema também são requeridos. As características de cada tipo são descritas a seguir:

4.1 Cold Standby

Cada tarefa usando este serviço é implementada por uma instância de tarefa padrão em Conic. O projetista especifica um conjunto de estaçoẽs aonde esta instância pode ser alocada. Inicialmente o sistema realiza todas as operaçoẽs de configuração necessárias

para alocar a instância na primeira estação definida no conjunto. Durante a operação do sistema, a estação onde a instância está residindo pode falhar. Após a deteção desta falha, uma nova instância daquela tarefa pode ser criada automáticamente na próxima estação especificada no conjunto associado a tarefa.

Este tipo de tarefa não preserva informação de estado entre duas ativaçoẽs. Um intervalo de tempo razoavel é necessario para que o sistema recrie uma nova instância da tarefa. O sistema assegura a criação da nova instância, contudo, atividades tais como a recuperação de estado e continuidade na cooperação com as outras instâncias de tarefas do sistema devem ser explicitamente programadas pelo projetista.

4.2 Hot Standby

Cada tarefa usando este serviço é implementada por duas instâncias idênticas, e pelo menos duas estaçoẽs são sempre necessárias. O projetista especifica um conjunto de estaçoẽs, o sistema aloca cada instância em uma estação diferente do conjunto. Cada instância está totalmente preparada para execução. Contudo, durante a operação do sistema elas possuem funçoẽs diferentes: Uma delas, chamada ativa, processa mensagens e coopera com as outras instâncias do sistema. A outra, chamada passiva, não executa nenhum processamento, a sua função é manter uma cópia atualizada do estado produzido pela instância ativa. Essa informação de estado é transferida automáticamente para a instância passiva. Inicialmente, o sistema cria e ativa as duas instâncias, assegurando que elas assumam funçoẽs diferentes. Durante a operação do sistema a instância ativa ou passiva pode falhar. No primeiro caso, a instância passiva assume o papel ativo e continua o processamento. No segundo caso, a instância ativa não é afetada pela falha. O sistema pode criar uma nova instância para substituir uma instância em falha; esta instância assume automáticamente o papel passivo.

Este tipo de tarefa preserva informação quando ocorre uma falha da instância ativa e a passiva assume o papel ativo. Considerando que a instância passiva está totalmente preparada, o intervalo de tempo tomado para reparar a tarefa é determinado pelo tempo necessário para detetar a falha e ativar a instância passiva. Está implicitamente assumido que uma falha dupla não ocorre. Depois da falha o sistema pode recriar uma nova instância re-estabelecendo o grau original de confiabilidade. Pode-se suportar mais que uma instância passiva, contudo isto requer mecanismos de suporte mais complicados. Na seção 5 será apresentada uma técnica especial, que independe da aplicação, e permite a recuperação transparente de falhas ao nível da programação e operação das tarefas.

4.3 Mecanismos de Suporte

Em Conic, a constituição do sistema da aplicação é especificada através de uma linguagem de configuração (ver apêndice). De maneira a suportar as tarefas tolerantes a falhas esta linguagem foi extendida de modo a permitir: (1) a especificação do tipo de redundância; e (2) as regras usadas para o controle dinâmico da criação de instâncias. Isto foi alcançado através da declaração abaixo:

```
CREATE  task_name:task_type, parameter_list, redundancy_type
        AT station_set;
```

A mesma estação pode aparecer em conjuntos de estações associados a tarefas diferentes. Neste caso é assumido que esta estação pode suportar somente uma das instâncias separadamente ou tôdas as instâncias ao mesmo tempo. Esta regra é aceitável para o controle de estruturas típicas em aplicações de controle. Contudo, maior flexibilidade pode ser obtida pela introdução de outras regras para o controle da configuração.

Durante a operação do sistema, dois módulos são usados no suporte das tarefas especiais (fig. 1). O Gerente da Configuração (GC) controla a configuração lógica com base em uma tabela gerada a partir da especificação do sistema; suas ações são colocadas em prática através de mensagens enviadas ao sistema operacional de cada estação, o qual suporta a capacidade de reconfiguração dinamica. O GC recebe informação de status gerada pelo Coletor de Status (CS); mudanças de status podem levar a que operações de reconfiguração sejam executadas. De modo a eliminar qualquer ponto singular de falha tanto o GC quanto o CS devem ser feitos tolerantes a falhas. Para o GC isto é conseguido através de sua implementação em hot standby. A obtenção de tolerância a falhas para o CS será discutida na seção 6.

Tarefas do tipo hot standby requerem mecanismos específicos para o suporte da recuperação de falhas, transferencias de estado, e determinação do papel desempenhado por cada instância (ver seção 6). A técnica usada para a recuperação de falhas será apresentada na proxima seção.

5. Técnica para Recuperação de Falhas

A obtenção da capacidade de tolerância a falhas basea-se na adição em Conic de uma primitiva especial de comunicação a qual extende a transação do tipo **pedido-resposta** (ver apêndice). Esta primitiva tem seu uso restrito a tarefas do tipo hot standby e a sua implementação garante que no caso da falha em uma instância, mesmo durante uma transação, esta completar-se-á transparentemente como se a falha não houvesse ocorrido. Isto é, o pedido será executado exatamente uma vez, e somente uma resposta será obtida, consequentemente, o estado da tarefa e do sistema será mantido consistente.

A figura 2 ilustra os fundamentos da técnica. A falha de qualquer uma das tarefas envolvidas pode ocorrer a qualquer momento, particularmente durante uma transação. Considerando que uma instância ativa falha, a instância passiva assume o papel ativo e continua a operação, podendo re-executar parte de seu programa neste processo. Em consequência da re-execução mensagens podem ser re-enviadas (pedido ou resposta); a semântica da transação é mantida pela simples filtragem de mensagens repetidas. Isto é realizado por um mecanismo cujo funcionamento é transparente ao nível das tarefas da aplicação (ver seção 6).

A transferência de estado, necessária para manter a atualidade do estado da instância passiva, e permitir a recuperação de falhas é feita automaticamente por um mecanismo padronizado. Esta capacidade é obtida a partir da especificação das portas que utilizam a primitiva especial de comunicação, as quais são denominadas portas especiais, e das duas regras abaixo:

R1: Uma transferência tem que ser executada após uma mensagem ser consumida por uma porta de entrada especial, e antes que qualquer resultado decorrente do processamento desta mensagem seja enviado para fora da tarefa.

R2: Uma transferência tem que ser executada antes que uma transação seja realizada por uma porta de saida especial, se uma transferência não houver sido executada desde a última transação realizada por esta porta.

Estas regras determinam quando a informação essencial para a sobrevivência do estado da tarefa e recuperação de falhas tem que ser transferida para a instância passiva. O conhecimento da identidade das portas especiais, definidas através da linguagem de configuração, permite que o código padronizado necessário ao cumprimento das regras seja inserido automáticamente nos programas das tarefas por meio de um pre-processador ou compilador. Desta forma, se as comunicaçoes forem restritas ao estilo pedido-resposta, as tarefas de um sistema podem ser programadas sem tolerância a falhas em mente, em uma fase posterior esta capacidade pode então ser obtida.

Uma primitiva especial: save, é disponível para especificar transferência de informação para a instância passiva. Sua implementação garante que a transferência é atómica mesmo em presença de falhas, e que a informação transferida é suficiente para permitir a re-execução da tarefa. Embora não seja requerido, a primitiva save é disponível ao nível da linguagem de programação. Isto permite o tratamento de situacoes particulares a cada aplicação. Como por exemplo: (1) portas especiais e normais podem ser usadas na mesma tarefa; (2) tarefas tipo cold standby e tipo hot standby podem ser interfaceadas.

A figura 3 mostra a aplicação das regras e a utilização da primitiva save. Considere que todas as transaçoes usam portas especiais, neste caso pode ser verificado que em qualquer situação de falha simples, a ação iniciada por req_1 completar-se-á

consistentemente sem a necessidade de qualquer save adicional. Por exemplo, considere que a tarefa_2 falha após executar a primitiva save. Na recuperação, ela executará novamente e gerará req_3, que já poderia ter sido gerada anteriormente na execução prévia. Se req_3 for uma duplicata, já consumida, ela é descartada; o mecanismo de suporte adquire a resposta correspondente: rep_3, e a envia para a tarefa_2. Em caso contrário, req_3 competirá normalmente para ser consumida na interface da tarefa_3. Tambem é possível que antes da falha, a rep_2 tenha sido gerada e consumida pela tarefa_1; na re-execução uma nova instância de rep_2 é gerada e será também filtrada pelo mecanismo de suporte.

6. Detalhes da Implementação

Dois mecanismos são requeridos para o suporte de tarefas em hot standby. O primeiro mecanismo suporta a execução das transaçoes especiais: filtragem de duplicatas e recuperação de mensagens resposta na área de dados da tarefa de entrada. Essas capacidades foram implementadas através de pequenas alteraçoes nas tarefas que suportam intercomunicação no sistema Conic básico. O segundo mecanismo suporta a execução da primitiva save e é implementado por tarefas padronizadas de gerenciamento, as quais são associadas a tarefas em hot standby ao nível da configuração. A invocação da primitiva save é transformada em uma transação pedido-resposta padronizada com a tarefa de gerenciamento, a qual adquire a informação a ser transferida da tarefa da aplicação e a transfere para a instância passiva. As estruturas de dados do núcleo do sistema Conic não precisam ser transferidas; elas são recuperadas pela re-execução da tarefa da aplicação durante a fase de recuperação. Em adição, a tarefa de gerenciamento inicializa o estado de novas instâncias, decide o papel a ser executado pela instância, e deteta falhas da outra instância do par.

No sistema descrito a deteção de falhas é realizada por duas entidades: o (1) Coletor de Status e as (2) Tarefas de Gerenciamento. Em ambos os casos um mecanismo baseado em timeouts o qual requer a geração e o recebimento periodico de mensagens é usado. Cada tarefa de gerenciamento tem a capacidade de detetar a falha da outra tarefa de gerenciamento (e da tarefa da aplicação) associada ao mesmo par. Isto permite a recuperação automatica e independente de falhas de instâncias de tarefas do tipo hot standby. Isto é útil em sistemas pequenos sem um gerente da configuração e para prover tolerância a falhas para o próprio gerente da configuração. De fato, a duplicação da capacidade de deteção a falhas pode ser evitada, contudo isto depende do sistema de comunicação particular. Por exemplo, um sistema de comunicação baseado em passagem de permissão (token) possue inerentemente a capacidade de deteção de falhas localmente a cada estação. Neste caso, deteção de falhas pode ser suprida por um único mecanismo. Correntemente, na PUC/RJ estamos implementando um sistema de comunicação, baseado em passagem de permissão, o qual será integrado ao nosso sistema.

7. Trabalhos Correlacionados

O sistema apresentado em [3] também usa processos duplicados para a obtenção de tolerância a falhas. Contudo, ele não provê uma técnica padronizada para recuperação de falhas, o que tem que ser feito caso a caso pelo programador. Em adição, a interface de entrada dos processos é definida por uma única fila de mensagens, o que não permite uma escolha não determinística das entradas nem o uso de guardas, como é possivel em Conic. Isto pode tornar a programação de algumas aplicaçoes diffcil.

Em [10] uma outra técnica para a construção de programas distribuídos tolerantes a falhas foi proposta. Ela requer a definição explícita de açoes tolerantes a falhas ao nível do programa, o que requer a intervenção cuidadosa do programador. Em adição, as comunicaçoes são realizadas através de variaveis compartilhadas. Isso não permite que as tarefas sejam projetadas independentemente, e é um conceito menos natural em um sistema distribuído onde mensagens tem que ser usadas em algum nível da implementação. Desta forma, o uso da técnica apresentada neste trabalho pode ser considerado vantajoso.

8. Conclusoes

Pela especificação do tipo de redundância requerido a capacidade de tolerancia a falhas para as tarefas é obtida automaticamente. A arquitetura é bastante simples o que contribue para que erros de projeto sejam menos prováveis. Os mecanismos de suporte são implementados por módulos escritos em Conic básico, o que facilita o seu transporte para qualquer tipo de processador já suportando essa arquitetura. Por outro lado os mecanismos específicos para o suporte de sistemas do tipo ADF são ortogonais aos mecanismos que suportam a capacidade de reconfiguração dinâmica (CS e GC), os quais são suficientes para o suporte de sistemas tipo BDF. Desta forma eles podem ser independentemente modificados e seletivamente usados para suprir as necessidades de cada aplicação particular. Finalmente deve ser ressaltado que, em princípio, para tarefas em hot standby, a tolerância a falhas pode ser obtida de modo transparente. Contudo, o sistema é flexível de modo a poder se adaptar a situaçoes específicas. Uma apresentação mais completa do sistema descrito neste trabalho, bem como dos detalhes de sua implementação pode ser encontrada em [7].

9. Apéndice

A linguagem de programaçcão Conic é baseada em Pascal, a qual foi extendida de maneira a prover interfaceamento através de portas e primitivas para troca de mensagens. O componente basico é o módulo tarefa (processo) que é equivalente a um programa Pascal sequencial, e pode ser separadamente projetado e compilado. A interface de uma tarefa é definida por um número de portas de entrada e saida. As primitivas de comunicação suportam dois tipos de transação: O tipo **pedido-resposta** (fig. 4), é projetado para uso

quando uma resposta, ou confirmação da aceitação, do pedido for requerida. Depois de enviar a mensagem, a tarefa fonte, tem a sua execução suspensa, sendo somente ativada após o recebimento da resposta. O tipo **notificação**, não prove qualquer segurança sobre o destino da mensagem de saida, a tarefa fonte continua executando após o envio da mensagem; esta transação permite flexibilidade.

A linguagem de configuração de Conic é usada para especificar um sistema (fig. 5) através da seleção e interconexão de várias instâncias de módulos. A declaração **USE** identifica o conjunto de tipos de módulos a ser usado na construção do sistema; a declaração **CREATE** especifica os nomes das instâncias a serem criadas no sistema; a declaração **LINK** especifica a interconexão das instâncias através da ligação de portas de saida com portas de entrada.

10. Referências

[1] Anderson, W., Lee, P., Fault Tolerance Principles and Practice, Prentice Hall International, 81.

[2] Andrews, D., Using Executable Assertions for Testing and Fault-Tolerance, 9th Symposium on Fault-Tolerant Computing, Junho 79.

[3] Bartlet, J., A Non-Stop Kernel, Proc. of the Eighth Symposium on Operating Systems Principles, Dezembro 81.

[4] Borg, A., et al, A Message System Supporting Fault Tolerance, Proceedings of the Ninth Symposium on Operating Systems Principles, Outubro 83.

[5] Kleinrock, K., et al, A Highly Reliable Distributed Loop Architecture, 10th Symposium on Fault-Tolerant Computing, Outubro 80.

[6] Kramer, J., et al., Conic: An Integrated Approach to Distributed Control Systems, IEE Proc., Vol. 130,Pt. E.N. 1, Janeiro 83.

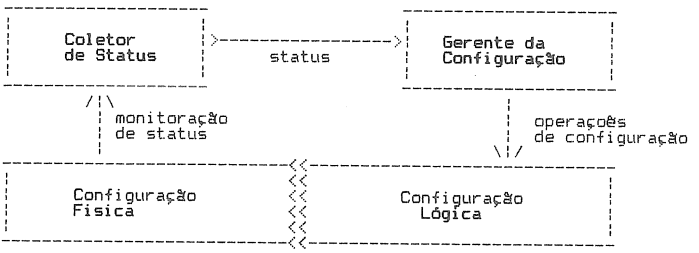
[7] Loques, O., G., A Fault Tolerant Distributed Computer Control System, Tese Ph.D, Imperial College, Londres, Fevereiro 84.

[8] Powell, R., Dependability Evaluation of Communication Support Systems for Local Area Distributed Computing, 12th Symposium on Fault-Tolerant Computing, Junho 82.

[9] Rennels, D., et al, A Study of Standard Building Blocks for the Design of Fault-Tolerant Distributed Computing Control Systems, 8th Symposium on Fault-Tolerant Computing, Junho 78.

[10] Scheneider, F. B., Schlichting, R., Towards Fault-Tolerant Process Control Software, 11th Symposium on Fault-Tolerant Computing, Junho 79.

[11] Wensley, H., et al., The Design and Analysis of a Fault Tolerant Computer for Aircraft Control, Proceedings of the IEEE, Outubro 78.



<< : Mapeamento Tarefas-->Estaçoës

Fig. 1 Funcionamento dos Mecanismos de Suporte

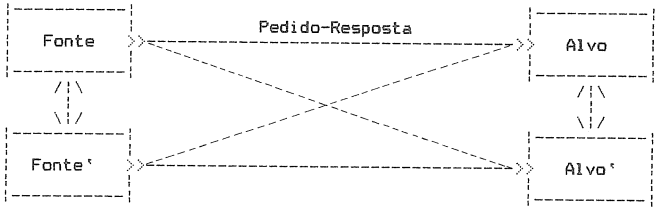


Fig. 2 Execução da Transação Especial

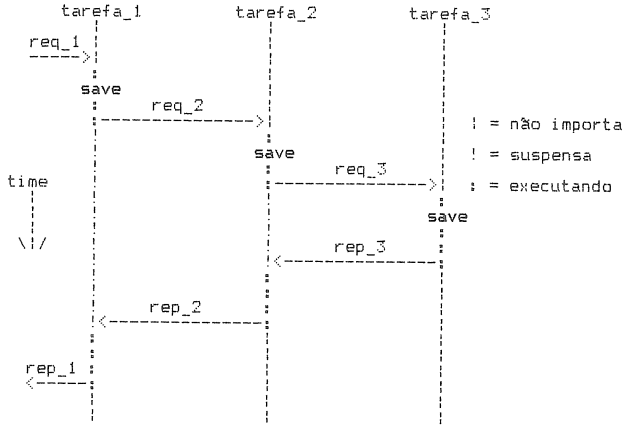


Fig. 3 Regras para uso da Save

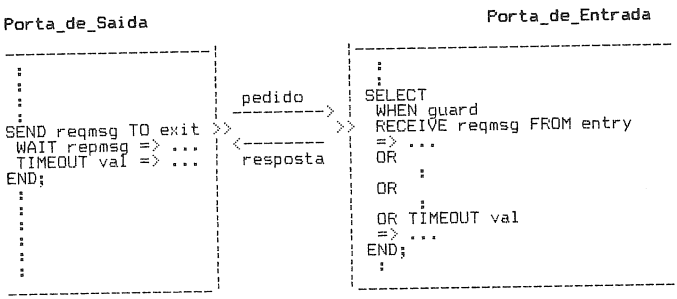


Fig. 4 Primitiva Pedido-Resposta

```
SYSTEM    patient_monitoring;
USE       bed_monitor, nurse_unit;
CONST     nbed = 4;
CREATE    bed[1..nbed]: bed_monitor;
          nurse: nurse_unit;
LINK      bed[1..nbed].alarms  TO  nurse.alarms;
          nurse.query[1..nbed]  TO  bed[1..nbed].status;
END;
```

Fig. 5 Exemplo de Especificação de Sistema