CONFIGURAÇÃO DINÂMICA NO SISTEMA ADES [1]

Wesley de Abreu[2], Joni Fraga[3] & Eraldo S. Silva[4]

Laboratório de Controle e Microinformática - LCMI - EEL
Universidade Federal de Santa Catarina - UFSC - CxP.476 88049 Florianópolis
SC Brasil
fone: (0482)319202 - fax: (0482)341524
E-mail lcmi@brufsc.bitnet

RESUMO

A linguagem LIS foi desenvolvida com o objetivo de facilitar a programação de aplicações distribuídas em tempo real. Esta linguagem encapsula vários aspectos da arquitetura do software, inclusive a configuração. Neste artigo são discutidos os principais mecanismos disponíveis nesta linguagem, para a configuração dinâmica de aplicações distribuídas. São abordados os princípios que orientaram o desenvolvimento de tais mecanismos e as possíveis implicações do processo de mudanças da configuração sobre a aplicação. Por fim, aspectos da implementação e os resultados obtidos são discutidos.

ABSTRACT

The LIS language was developed to facilitate the real-time distributed applications programming. This language encapsules many aspects of software architecture, including the configuration. In this paper, the main available mechanisms of the language for dynamic configuration of distributed applications are discussed. Also, the principles that oriented such mechanisms development and some implications of the configuration change process on the application are presented. Then, some implementation aspects and the results are described.

---

[1] Projeto Financiado pela Fundação Banco do Brasil (FBB)

[2] Engenheiro Eletricista (UFSC, 1986), mestre em Engenharia Elétrica - UFSC; linguagens de programação distribuída e aplicações em tempo real; professor Dpto. Ciências da Computação UDESC- FEJ.

[3] Engenheiro Eletrônico (UFRGS, 1974), mestre em Engenharia Elétrica (UFSC, 1979), Dr (LAAS-INPT, 1984); sistemas distribuídos, tolerância a falhas e aplicações em tempo real; professor do Dpto. de Engenharia Elétrica UFSC.

(4) Engenheiro Eletricista (UFSC, 1985), mestre em Engenharia Elétrica (UFSC,1988); sistemas distribuídos, aplicações em tempo real; Engenheiro no LCMI-UFSC.

1. INTRODUÇÃO

A introdução de mecanismos que tornam um Sistema Informático Distribuído flexível, especialmente aqueles com aplicações em Controle e Automação, é direcionada por três motivos básicos. Em primeiro lugar, a própria aplicação pode exigir que o sistema se altere dinamicamente seja por criação (ou destruição) dinâmica de tarefas seja pela criação de novos canais de comunicação (ex.centrais telefônicas). Num segundo ponto, a evolução tecnológica, o crescimento ou alterações da planta controlada (devido a expansão e/ou a manutenção desta), fazem com que haja necessidades de reconfiguração do sistema de uma forma segura e rápida, preferencialmente com o sistema em curso de operação. Finalmente, a flexibilidade se impõe em sistemas confiáveis onde a presença de elementos faltosos não represente a ruptura de funções no sistema. Neste caso, as propriedades de flexibilidade são importantes para a definição e manuseio de redundâncias quando da implementação de técnicas de Tolerância a Faltas.

Embora em alguns sistemas operacionais sejam fornecidos recursos para configuração dinâmica (ex. Chorus [Gien, 89]), somente a abordagem de construção de sistemas distribuídos a partir de linguagens tem mostrado simplicidade e facilidade no gerenciamento de configuração de uma aplicação. Nesta linguagem de programação distribuída, a separação dos aspectos de configuração (estruturais) da programação dos componentes de uma aplicação, para fins de configuração dinâmica tem sido consenso geral das pesquisas realizadas neste domínio [Kramer, 85].

A Linguagem de Implementação de Sistemas-LIS, apresentada neste artigo, faz parte de um ambiente para o desenvolvimento e execução de software distribuído de tempo real (ADES) ,(Fraga,89), que esta sendo desenvolvido no LCMI-EEL-UFSC. Esta linguagem segue o paradigma proposto no sistema CONIC [Magee, 87] para a configuração de sistemas com algumas diferenças fundamentais. Ao contrário deste último, cujas unidades de configuração constituem-se em estações (lógicas), na LIS tais unidades são componentes de software (módulos) que encapsulam processos. Desta forma torna-se factível a realização de configurações dinâmicas sem a necessidade de reconstruir toda uma estação.

Este artigo tem como objetivo apresentar os principais aspectos dos mecanismos para configuração dinâmica suportados pela linguagem LIS . Na seção 2 discute-se os requisitos para a configuração dinâmica referenciando-se para tanto o modelo proposto por [Krammer, 88] de onde são extraídas algumas regras gerais para a manutenção da estabilidade do sistema. Na seção 3 a linguagem LIS é descrita destacando-se aquelas características relacionadas à configuração dinâmica que a tornam compatível com as propriedades verificadas na seção 2. A seção 4 descreve o processo de configuração dinâmica com alguns detalhes da implementação do Gerenciador de Configuração. Em 5 discutimos a eficácia da LIS tendo em vista as condições colocadas em 2. Os resultados concretos da implementação dos mecanismos são discutidos na seção 6.

A linguagem LIS com o suporte de configuração, nos vários aspectos levantados neste texto está implementada na forma de um protótipo já testado em simulações, exemplos e utilizada em ensino.

2. CONFIGURAÇÃO DINÂMICA: REQUISITOS

O problema fundamental com a configuração dinâmica em sistemas distribuídos é aquele relacionado com a manutenção da estabilidade do sistema durante o processo de mudanças. A estabilidade depende essencialmente de um modelo de gerenciamento de mudanças a ser utilizado na passagem de um estado de configuração para outro. Sob o ponto de vista da relação ambiente/sistema, o Gerenciamento de Configuração deve assegurar as seguintes condições:

- as declarações de mudanças devem ser independentes dos algoritmos, protocolos e estados da aplicação; o que permite um gerenciamento de configuração genérico e independente da aplicação;

- mudanças devem deixar o sistema em um estado consistente;

- mudanças devem minimizar as rupturas nas aplicações. A ordem de grandeza do intervalo de transitório das partes em mudança não deve afetar o comportamento lógico-temporal do ambiente diretamente envolvido. O gerenciamento de configuração deve atuar nas partes do sistema envolvidas nos processos de mudança. O resto do sistema deve continuar sua execução normalmente.

O gerenciamento de configuração, afim de atender as duas últimas condições mencionadas acima, deve levar em consideração as influências causadas em componentes (módulos) dependentes direta ou indiretamente daqueles envolvidos nas mudanças. Neste sentido, o suporte de configuração deve ter uma interface com a aplicação para poder determinar um estado apropriado desta para a execução da configuração dinâmica. Esta interface, para estar de acordo com os requisitos acima deve ser genérica e independente de uma aplicação em particular.

O processo de mudanças deve apresentar os módulos envolvidos nestas operações em estado de "quiescência". Considerando que as interações entre módulos se dão em transações, a quiescência de um módulo implica em:

- o módulo não se encontra engajado em transações iniciadas por ele;

- o módulo não iniciará novas transações; e

- nenhuma transação iniciada por outro módulo, terá a participação de um módulo quiescente .

Com a quiescência dos módulos envolvidos na mudança, as operações para tal podem ocorrer de forma que o estado do sistema permanecerá consistente, ou seja, os módulos em processo de mudança não implicarão no surgimento de transações incompletas na aplicação. Para um módulo atingir a quiescência, este deve passar inicialmente por um estado de passividade com a propriedade de não iniciar transações,

mas somente responder possíveis transações iniciadas por outros módulos. A quiescência só é atingida quando o conjunto de módulos que possam iniciar transações com este módulo estiverem em estado passivo.

3. A CONFIGURAÇÃO DINÂMICA NA LINGUAGEM LIS

3.1. LIS : Uma Linguagem de Programação Distribuída

A linguagem LIS, foi desenvolvida com o propósito de facilitar a construção e a manutenção de aplicações distribuídas, especialmente aquelas que envolvem atividades relacionadas ao controle de processos e automação industrial.

A LIS implementa um modelo de programação que torna independente a programação de unidades de software (módulos) da configuração do sistema. Esta abordagem é classicamente conhecida como a programação em pequena escala e a programação em larga escala [DeRemer, 76].

Neste sentido a LIS é dividida em duas sub-linguagens: a linguagem LINCE de programação de componentes e a linguagem LINCS de configuração do sistema. Um componente (módulo) é visível externamente, de maneira única, através de portos tipados de saída e entrada. Todas as referências internas do módulo, com fins de comunicação exterior, são realizadas através de mensagens enviadas/recebidas dos portos de módulo, o que garante uma total independência dos demais módulos do sistema.

As unidades de concorrência do sistema ("tasks") são definidas internamente ao módulo e não são manipuladas pela linguagem de configuração LINCS. A figura 1 apresenta o modelo de programação incorporado na linguagem LIS.

A programação de uma aplicação distribuída através da LIS é sistemática e tipicamente realizada da seguinte forma: inicialmente uma biblioteca de tipos módulo é formada a partir da construção de cada módulo pela LINCE. A seguir, e opcionalmente, sub-sistemas podem ser compostos por um grupo dos módulos da biblioteca (declaração GROUP MODULE) e finalmente o sistema é construído segundo uma declaração SYSTEM. As declarações GROUP MODULE e SYSTEM são descritas em LINCS. Na declaração SYSTEM são definidos todos os tipos módulos e grupos a serem utilizados (definição de contexto), as criações de instâncias destes tipos são realizadas em estações específicas, e finalmente todas as interfaces (portos) são conectados após uma devida verificação de tipos relacionados. Nas figuras 2a, 2b e 2c são apresentados as declarações de tipo módulo, grupo e sistema, respectivamente.

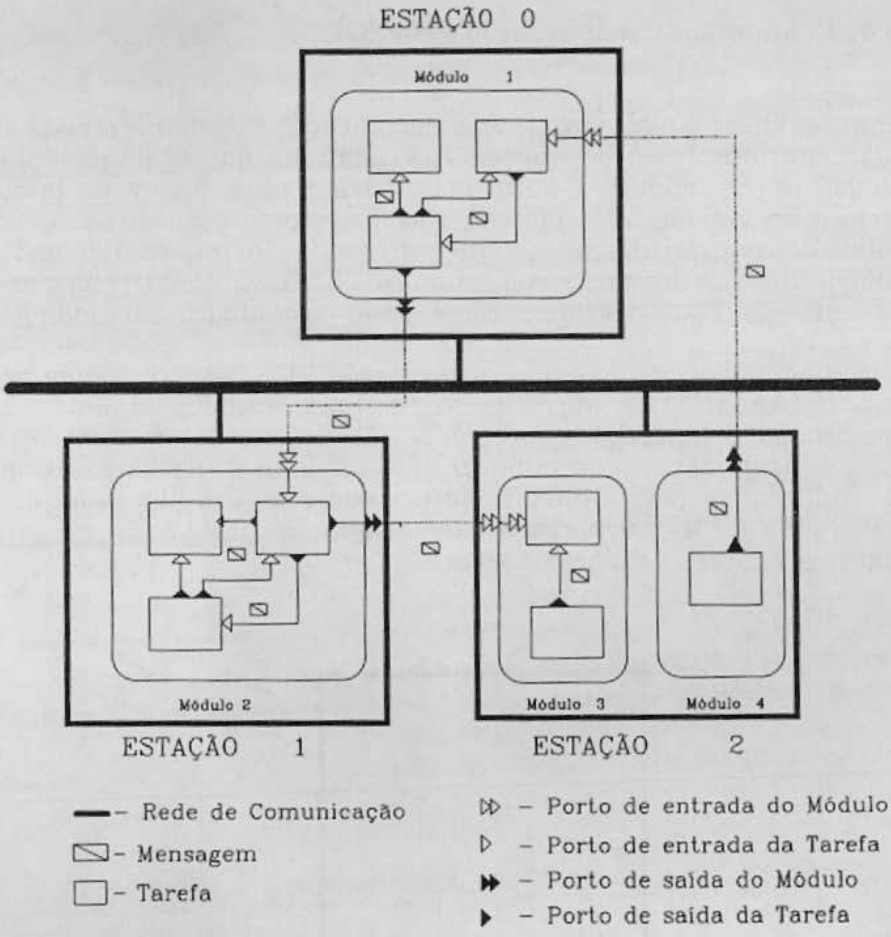

Figura 1. Paradigma de Programação

```
Module nome:
Port
 < portos do modulo>
Task nome;
Port
 < portos da tarefa>
Begin
  < corpo da tarefa>
End;
Link
  < ligacao portos da ta-
    refa e modulo>
Endmodule
```

(a) TIPO MODULO(LINCE)

```
Group module;
Use
  < Define contexto >
Port
  < portos do grupo>
Create:
< cria instancias dos
   tipos modulos >
Link
 <ligacao entre portos
   do grupo e instancias>
End.
```

(b) DECLARACAO DE GRUPO (LINCS)

```
System nome:
Use
  < Define contexto >
Create at Station < >:
< define instancias a
   serem criadas  na
        estacao    >
Link
  < ligacao  entre os
    portos da  ins -
        tancias      >
End.
```

(c) DECLARACAO SYSTEM (LINCS)

Figura 2. Declarações em LINCE e LINCS

3.2. Operações de Configuração: a declaração CHANGE

A estrutura modular e a hierarquização da aplicação segundo níveis de abstração são propriedades intrínsecas à linguagem LIS. Em adição, a impossibilidade de referências diretas de um módulo a outro componente proporciona propriedades ao sistema que viabilizam a evolução e manutenção do sistema em execução, através de operações 'on-line' de conexão/desconexão de portos, criação/remoção de instâncias de módulos, parada/partida de instâncias de módulos e carregamento/remoção de tipos módulos em uma estação. Estas operações caracterizam a configuração dinâmica.

Para permitir mudanças dinâmicas, a linguagem LINCS suporta uma declaração de configuração CHANGE, que apresenta a mesma estrutura de uma declaração SYSTEM, acrescida de "operações inversas". As operações inversas possibilitam desfazer ligações, destruir instâncias e remover tipos módulos do sistema distribuído. As mudanças são controladas pelo Módulo Gerenciador de Configuração do sistema operacional distribuído (SOD), a partir destas declarações CHANGE (figura 3). O SOD é discutido com maiores detalhes na seção 4.

```
Change nome;
Use <define contexto>
Create at Station < >::
    < define instancias>
Delete< instancias >
Link  < define portos>
Unlink < define portos >
Remove < define contexto>
Start <instancia>
Stop < define instancia>
END.
```

Figura 3. Declaração CHANGE

3.3. Mecanismos de Manutenção de Consistência

O problema de inconsistência está relacionado com as transações existentes entre os módulos que compõe o sistema. Um módulo pode iniciar transações (emissor) ou pode ser sensibilizado por transações (receptor). Quando um módulo é envolvido nas mudanças, o efeito da retirada deste da configuração pode ser sentida nos módulos que iniciaram transações e/ou nos módulos receptores. Estes efeitos estão diretamente ligados a produção de inconsistências no sistema. A LIS fornece alguns mecanismos para o tratamento destas inconsistência", discutidos a seguir.

Tratadores de Exceção na Comunicação

As inconsistências podem ser reduzidas a partir das comunicações através das cláusulas de esgotamento de tempo e de falha de ligação de portos introduzidas na sub-linguagem LINCE (FAILTIME E FAILINK). As exceções causadas pela falta de resposta a uma transação iniciada pode ser detectada pelo esgotamento de tempo "timeout" o que permite o tratamento do "não envio" de mensagens. A cláusula de tempo permite a programação de um manuseador para exceções do tipo esgotamento de tempo em envios síncronos ou em operações de recepção em tarefas. Na segunda cláusula é permitido, na tarefa emissora, um manuseador de exceções "falta de ligação" (em envios síncronos e assíncronos). Estes mecanismos minimizam principalmente os efeitos no módulo iniciador da transação quando o outro participante, o módulo que recebe o pedido da transação, está envolvido no processo de mudanças.

Pontos de Sincronismo Estendido

No processo de retirada na configuração, módulos que iniciam transações (envio de mensagens), podem gerar processamentos "órfãos" nos receptores. No sistema CONIC [Magee, 87], para tentar evitar o início de transações quando do envolvimento do emissor em mudanças, foi introduzida a noção de "pontos de sincronismo". Estes pontos servem como meios de sinalização entre o suporte de configuração dinâmica, e os módulos da aplicação, de modo que operações de configuração nestes últimos só venham a ser executados entre transações, ou seja, antes de um módulo começar uma nova transação através do envio.

No sistema ADES, inicialmente os pontos de sincronismo são colocados estrategicamente em cada tarefa do módulo considerado, através da primitiva SINC definida na linguagem LINCE.

Os pontos de sincronismo, envolvem a suspensão implícita das tarefas, quando em processo de configuração, antes que estas iniciem uma nova transação. Isto contribui para diminuir as possíveis inconsistências no sistema informático, mas não oferece ainda para o engenheiro do processo (ou programador) meios para minimizar os efeitos destas inconsistências sobre a aplicação. Diante disto, o mecanismo ponto de sincronismo foi estendido no sistema ADES de modo a permitir que o programador, considerando a importância de determinados módulos para a aplicação, possa programar um conjunto de operações que devem ser executados pelas tarefas quando em estado de configuração, antes de se auto-suspenderem. A primitiva SINC toma então a seguinte sintaxe quando introduzida em uma tarefa:

```
decl_sinc ::= SINC
                    instrução [";"instrução]*
              ENDSINC
instrução ::= instrução_extensão/instrução_pascal
```

O programador considerando as particularidades da aplicação, poderá então se valer deste mecanismo e preparar a aplicação para mudanças eventuais envolvendo determinados módulos. A colocação ou não da primitiva SINC é critério do programador do módulo. A não colocação implica na parada imediata do módulo durante a configuração dinâmica do sistema.

O suporte de configuração do ADES foi projetado para admitir a execução da declaração CHANGE com ou sem esta sincronização, ficando portanto a escolha a critério do programador de configuração. O processo de configuração sem sincronismo é sempre utilizado quando da impossibilidade de continuar a execução de uma declaração CHANGE, a não interação entre o suporte de configuração e a aplicação possibilita um retorno mais rápido ao estado de configuração anterior.

3.4. Reconfiguração interna dos módulos

O módulo como unidade de configuração tem a sua estrutura interna não visível pelo configurador. A configuração interna é determinada através de declarações de ligações (LINK) da linguagem de componentes (LINCE). Uma particularidade propiciada pelo modelo de programação é a modificação destes "LINKs" internos, dinamicamente, a partir das tarefas do próprio módulo. A instância do módulo pode ter sua configuração interna modificada em tempo real ou segundo exceções; para tanto, por exemplo, basta ter uma tarefa alternativa em estado de suspensa, não participando da configuração interna. A inclusão desta pode ser programada (através de "UNLINK's" e "LINK's") a partir de outra tarefa do módulo.

Este tipo de mecanismo é extremamente útil no sentido de permitir mudanças (internas à instância) executadas a partir da aplicação, sem envolver o suporte de configuração.

4.PROCESSO DE CONFIGURAÇÃO DINÂMICA

Execução das Mudanças

A Configuração Dinâmica envolve a definição de módulos gerenciadores do SOD, que concorrem com módulos da aplicação nas estações. Estes gerenciadores são responsáveis pelas operações sobre módulos, ligações de portos e acesso remoto à memória, completando então o suporte para configuração dinâmica (figura 4).

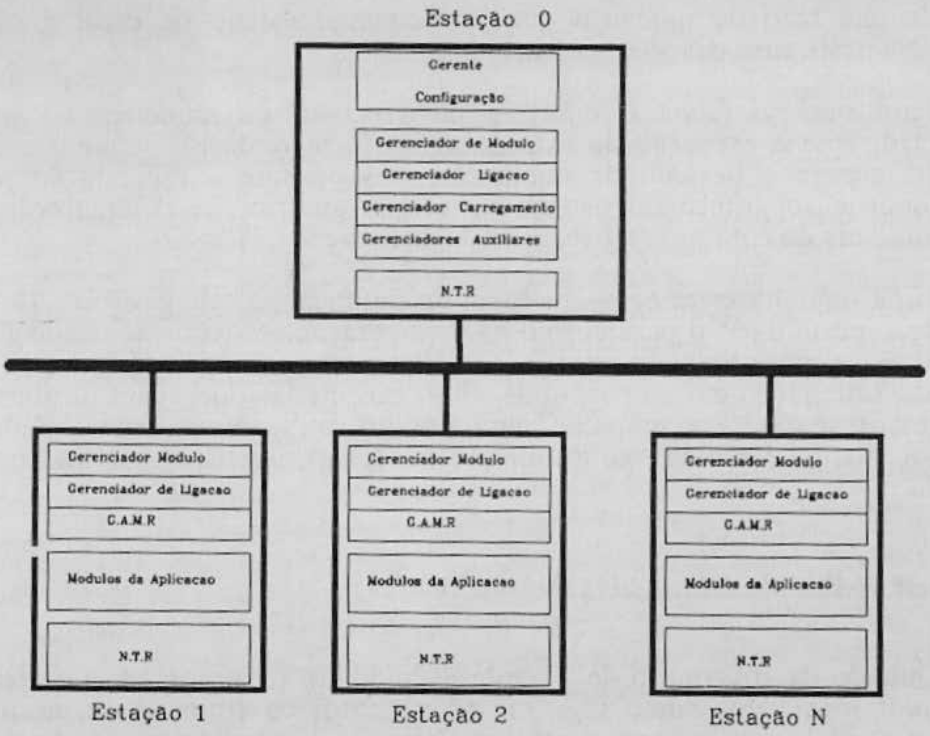

Figura 4. Suporte para Configuração Dinâmica

A tradução de uma especificação CHANGE gera uma Base de Dados e uma Lista de Ações. A Base de Dados representa uma versão (temporária) de configuração do sistema, incorporando as mudanças especificadas. A Lista de Ações gerada será interpretada pelo Módulo Gerenciador de Configuração.

A Lista de Ações deve traduzir as necessidades de operações para provocar a mudança. Estas operações serão executadas a partir de mensagens enviadas pelo Módulo Gerenciador de Configuração que controla a execução da lista, aos módulos gerenciadores das Estações de Trabalho e de Execução. Se todas as ações forem realizadas com sucesso, a representação temporária do sistema será efetivada como a versão atual do "estado de configuração do sistema", podendo ser iniciado novas modificações no sistema.

Paralelamente à execução destas operações de mudança é gerada uma lista de ações reversa que contêm declarações de comandos com semântica inversa às operações na lista de ações. Na ocorrência de qualquer exceção que impeça a concretização de mudança é gerado uma inconsistência no sistema que não representa um estado de configuração conhecido. Em visto disto, o sistema deve retornar à configuração anterior. Este retorno é realizado através da execução dos comandos da lista de ações reversa a

partir da operação de mudança que introduziu o estado de erro; a representação temporária neste caso deve ser descartada.

Para que se tenha o controle do processo de mudanças, a princípio, foi estabelecido que a execução da lista de ações fosse realizada seqüencialmente. Esta estratégia alonga o período de mudanças, mas permite a recondução com extrema simplicidade e segurança do sistema ao estado anterior de configuração quando da impossibilidade de continuar a execução da declaração CHANGE.

Uma segunda estratégia, em curso de definição, estabelece uma Lista de Ações por estação permitindo o paralelismo na configuração. A execução destas listas deve se dar igualmente, para todas as estações afetadas pela declaração CHANGE, respeitando uma certa ordenação em etapas, onde serão executadas operações do mesmo tipo, ou seja, primeiro se dará uma etapa de "para módulo", na seguinte serão desfeitas ligações e assim por diante. Uma vez confirmada a execução das listas a nova configuração é assumida.

Operações de Parada com pontos de sincronismo

Quando da solicitação de "parada de módulo" (operação da Lista de Ações), o gerenciador local deve então registrar no descritor de controle da instância do tipo módulo envolvido que o mesmo está em "estado de configuração". Após o registro o gerenciador se bloqueia. Por sua vez, cada tarefa do módulo, ao executar a primitiva SINC verifica se o mesmo está em estado de configuração; no caso de existir confirmação, a tarefa é suspensa.

Considerando a natureza multitarefa dos módulos na linguagem LIS, só ocorrerá uma "parada" de módulo quando todas as suas tarefas forem suspensas. A troca de prioridades das tarefas para níveis mais altos é uma providência a ser tomada pelo gerenciador de modo a acelerar esta operação de parada. Na continuidade desta operação, quando a última tarefa do módulo for suspensa, o gerenciador local será ativado, dando prosseguimento na execução de outras operações da lista de ações no processo de mudança.

5. DISCUSSÃO

A análise do estado global do sistema, é importante no sentido de definir a estratégia da realização das modificações. Em um determinado sistema, podemos ter relações de dependência variadas entre os módulos que compõe uma determinada aplicação. A dependência entre módulos é uma grandeza de difícil quantificação, mas de um modo geral ela pode ser classificada em **Alta, Média e Baixa dependência:**

- **Alta Dependência** : a verificação da possível estabilidade do sistema em função de operações de mudança sobre um determinado módulo é complexa. A identificação de um estado do sistema apropriado para iniciação do processo de mudanças nem sempre é possível; com isto

existe uma alta probabilidade da propagação dos efeitos resultantes das mudanças sobre o sistema. Nestes casos é necessária a parada do sistema para que nova configuração seja realizada estaticamente.

- **Média Dependência** : existe uma região crítica do sistema que pode ser identificada facilmente como área de propagação dos efeitos do processo de mudança. Determinada esta área, pode-se parar os módulos que provocariam início de transações no módulo envolvido nas mudanças, gerando desta forma uma região que permite a quiescência do módulo; isto caracteriza uma parte do sistema em contínua operação e outra em estado de configuração.

- **Baixa Dependência** : operações de mudança sobre um determinado módulo não se propagam sobre o restante do sistema na forma de inconsistência (módulos independentes).

Na elaboração de uma especificação CHANGE, o programador deve portanto levar em conta a análise de dependências entre os módulos. O conhecimento gerado desta análise adicionado às possibilidades de sincronização entre o suporte de configuração e os módulos da aplicação e ainda de preparar a retirada de módulos devem atender os requisitos colocados no item 2, no sentido de manter o sistema estável no processo de mudança.

6. RESULTADOS E CONCLUSÃO

Os mecanismos de configuração dinâmica descritos neste artigo foram todos implementados no protótipo ADES, inclusive o gerenciamento de configuração distribuído. A estação de desenvolvimento bem como as estações de execução utilizadas são PC-compatíveis, sendo que no caso da primeira esta sendo utilizado o DOS-compatível como hospedeiro, o que limita a edição e a compilação "on-line" de novos tipo módulos e de novas configurações de mudança. No entanto, pode-se realizar a configuração dinâmica gerando-se previamente as listas de ações, antes da realização da configuração estática do sistema (configurações pré-planejadas).

No momento está-se estudando um modelo de execução da LIS sobre o UNIX de maneira a permitir a configuração dinâmica de módulos dentro de estações (lógicas) a serem executadas em uma "WorkStation".

A reconfiguração interna dos módulos está sendo utilizada no sistema ADES para a implementação dos conceitos "bloco de recobrimento" [Anderson, 81] e "bloco de recobrimento distribuído" [Kim, 88]. Está em estudo a extensão da linguagem LIS, para suportar estes conceitos de modo a simplificar a programação de redundâncias, visando a tolerância a faltas de projeto e físicas em aplicações em tempo real [Rodrigues, 89].

BIBLIOGRAFIA

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

[DeRemer, 76]: DeRemer, F & Kron, H.H. Programming-in-the-Large versus Programming-in-the-Small. IEEE Trans. on Software Engineering, SE-2 (2), Juin 1976.

[Fraga, 89]: Fraga, J.S., Farines, J.M, Abreu, W.M.B., Nacamura Jr., L., Coelho Filho, O. "ADES: Ambiente de Desenvolvimento e Execução de Software Distribuído", Actes Du Séminaire Franco-Brésilien Sur Les Systèmes Informatiques Répartis, Florianópolis-SC-Brasil, September 89.

[Gien, 89]: Gien, M., Architecture des Rystèmes Répartis - Une Nouvelle Génération d'UNIX, Actes Du Séminaire Franco-Brésilien Sur Les Systèmes Informatiques Répartis, Florianópolis-SC-Brasil, September 89.

[Kim, 88]: Kim, K.H. & Yoon, J.C. Approaches to Implementation of a Repairable Distributed Recovery Block Scheme. 18th International Symposium On Fault Tolerant Computing, Pittsburg, Juin 1988.

[Krammer, 85]: Krammer J., Magee J., "Dynamic configuration for distributed systems " IEEE Transactions on Software Engineering, April 1985.

[Krammer, 88]: Krammer J., Magee J., A Model for Change Management IEEE Distributed Computing Systems in the '90s, Hong Kong, September 1988.

[Magee, 87]: Magee J., Krammer J., Sloman, M., Constructing Distributed Systms in CONIC. Imperial College Research Report Doc 87/4, March 1987.

[Rodrigues, 89]: Rodrigues, V. "Modelos para Programação Distribuída Tolerante à Falhas". Nota interna LCMI-UFSC, Florianópolis, agosto 89.