*SBA:* Controle & Automação, Vol. 1, Nº 4, pp. 264-276

# TENDÊNCIAS DA PESQUISA EM SISTEMAS DISTRIBUÍDOS

Yves Deswarte

INRIA - LAAS/CNRS
7, Avenue du Colonel Roche
31Ø77 - Toulouse - FRANÇA

Resumo

Este trabalho visa descrever sucintamente as principais orientações da pesquisa sobre sistemas distribuídos e levantar tendências mais significativas para os próximos anos. São apresentadas as necessidades de suporte para aplicações distribuídas e os requisitos de qualidade destes serviços. O ciclo de vida de uma aplicação e as particularidades dos sistemas distribuídos em automação industrial são também objeto desta publicação.

Research Trends in Distributed Systems

Abstract

In this paper are briefly described the principal orientations on distributed systems researchs and the more relevant trends for next years. Support and quality requirements for distributed applications are also presented. At last, life cycle of such application and distributed system particularities in industrial manufacturing systems and process control are also addressed.

## 1. INTRODUÇÃO

Os Sistemas Distribuídos são, desde alguns anos, o ponto de encontro de algumas disciplinas entre as mais ativas da informática: comunicações, redes, sistemas operacionais, linguagens, base de dados, controle de processos, modelização, tolerância a faltas, etc. Este fenômeno acompanha o desenvolvimento do mercado da informática distribuída que atinge um crescimento mundial da ordem de 75% ao ano (enquanto o mercado global da informática cresce somente 15% ao ano).

Este artigo visa descrever sucintamente as principais orientações da pesquisa sobre os sistemas distribuídos e levantar as tendências mais significativas para os próximos anos.

Cada vez que for possível, serão referenciados projetos de pesquisa ilustrando estas tendências, principalmente na França e na Europa (que o autor conhece melhor).

A primeira parte deste artigo apresenta um breve histórico do desenvolvimento dos sistemas distribuídos e tenta levantar as razões e as direções futuras.

A segunda parte é dedicada aos suportes fornecidos às aplicações distribuídas: comunicações, modelos de aplicação e sistemas operacionais.

A terceira parte apresenta os serviços esperados do sistema distribuído, em particular no que diz respeito à transparência da distribuição e à segurança de funcionamento.

A quarta parte descreve o ciclo de vida de uma aplicação distribuída e as ferramentas e métodos que lhe são associados: concepção, realização, operação.

Por fim, a última parte aborda certas particularidades dos sistemas distribuídos de automação industrial.

## 2. HISTÓRICO DOS SISTEMAS DISTRIBUÍDOS

### De 196Ø a 197Ø

Com o desenvolvimento dos sistemas a tempo compartilhado ("time-sharing"), o afastamento dos periféricos conduz ao refinamento de técnicas de comunicação ponto-a-ponto e multi-ponto (concentrador de terminais) ao redor do computador central, bem como ao gerenciamento de periféricos distantes ("remote batch processing").

Paralelamente, desenvolveram-se os primeiros sistemas distribuídos com vários computadores interligados por conexões ponto-a-ponto, para aplicações muito específicas (Sistema SAGE de vigilância do território dos Estados Unidos, sistema SABRE de reserva de passagens aéreas). Os protocolos implementados são muito simples e gerenciados diretamente pela aplicação.

### De 197Ø a 1975

Para reduzir o custo das comunicações, são concebidas redes gerais de grande distância, fornecendo um serviço de comunicação independente da aplicação: nos Estados Unidos: ARPANET, na França: CYCLADES e depois TRANSPAC. Estas redes necessitam de protocolos mais complexos (comutação de pacotes, roteamento, etc.) mas dão lugar a novos serviços gerais (transferência de arquivos).

### De 1975 a 198Ø

Estes anos vêem o surgimento das primeiras redes locais e o desenvolvimento das estações de trabalho com compartilhamento de recursos (arquivos, impressoras).

Por outro lado, os principais construtores desenvolvem suas próprias redes homogêneas (fechadas = "proprietárias"): na IBM: SNA; na Honeywell-Bull: DSA; na DEC: DNA, etc.

### De 198Ø a 1985

Assiste-se ao aparecimento dos primeiros sistemas operacionais distribuídos por interconexão de sistemas homogêneos (geralmente UNIX) com sistemas de gerenciamento de arquivos distribuídos (Ex: Newcastle Connection (Brownbridge et alii, 1982)) ou permitindo a execução à distância (Ex: LOCUS (Popek & Walker, 1985)).

Por outro lado, a importância dos organismos de normalização cresce : modelo de referência das interconexões de sistemas abertos (Open System Interconnection Reference Model) da ISO (Zimmermann, 198Ø) que permitem a comunicação do hardware (e do software) de vários fornecedores.

Aparecem também as "redes digitais a integração de serviços" (ISDN: Integrated Service Digital Networks).

### Tendência Atual

A tendência geral está na abertura:

- abertura dos fabricantes que tentam aproximar suas arquiteturas de redes aos padrões ISO ou aos padrões "de fato" (TCP/iP, Ethernet) ou ainda desenvolvem "gateways" para a interconexão de redes heterogêneas (Ex: DSA-SNA);

- abertura em direção de sistemas operacionais "não-proprietários"; por exemplo, os trabalhos de normalização do UNIX pelo grupo X-Open.

Mas os trabalhos de normalização são muito lentos: são necessários vários anos para obter uma norma aceitável. Para ganhar tempo, os organismos de normalização têm tendência a normalizar padrões "de fato" pré-existentes: por exemplo, Ethernet, 1Ø anos após seu desenvolvimento, foi normalizado pela IEEE (IEEE 8Ø2.3) e depois pela ISO (ISO 88Ø2). Além disso, a multiplicidade das necessidades dos usuários e das tecnologias dos fabricantes conduz a uma multiplicação das "classes", o que leva a normas não-homogêneas. Para remediar este

fato, "grandes" usuários tentam impor sub-conjuntos destas normas, adaptados a suas necessidades: por exemplo, General Motors propõe MAP (GM, 1986), British Aerospace associou-se a BMW, Aeritalia e a outros usuários no projeto CNMA do programa europeu ESPRIT (CNMA, 1986).

Orientações Futuras

É possível prever que, a mais ou menos longo prazo, os sistemas distribuídos integrarão aplicações distribuídas múltiplas. Desta forma, sobre a(s) rede(s) de uma mesma empresa coabitarão e cooperarão:

- aplicações de controle de processos, de supervisão, de segurança;
- aplicações de gerenciamento de estoques, de planificação;
- aplicações de gestão administrativa (comandos, faturas, pessoal);
- aplicações de estudo e desenvolvimento (CAD);
- aplicações de relações públicas;
- aplicações de relações internas e externas (correio eletrônico, ISDN, acesso a banco de dados, transferência eletrônica de dinheiro, etc.).

## 3. A PESQUISA SOBRE OS SUPORTES DE APLICAÇÕES DISTRIBUÍDAS

### 3.1. Comunicação

A pesquisa nesta área é menos dinâmica que há alguns anos atrás:

- os problemas básicos das comunicações (qualidade do serviço, controle de fluxo, etc.) foram solucionados de forma satisfatória:
- os métodos e ferramentas de concepção existem e são normalmente utilizados, pelo menos para as camadas baixas dos protocolos:
- as normas existentes são um freio à evolução.

Entretanto, pesquisas continuam em áreas como:

- comunicações com taxa de transferência muito alta (várias Gbits/s);
- confidencialidade das comunicações (criptografia, fragmentação-dissemi-nação);
- utilização da difusão fornecida pelas camadas baixas dos protocolos.

### 3.2. Modelos de Aplicação Distribuída

Conceber uma aplicação distribuída consiste em imaginar e depois estruturar um conjunto de ações ocorrendo em paralelo, em cooperação e/ou concorrência sobre computadores geograficamente distribuídos.

Esta estruturação é facilitada pela utilização de modelos. Classificaremos a seguir os modelos em quatro categorias, ainda que certos modelos propostos possam pertencer a várias categorias ao mesmo tempo.

#### 3.2.1. Modelo Cliente-Servidor

É um dos primeiros modelos propostos para os sistemas distribuídos, em particular para o compartilhamento dos recursos (servidores de arquivos, servidores de impressora, etc.). Um servidor pode ser centralizado ou composto de várias entidades equivalentes ou ainda ser, ele mesmo, um cliente de outro servidor (Svoboda, 1984).

#### 3.2.2. Modelo Transacional

Este modelo foi inicialmente definido para os acessos múltiplos a bases de dados (ex.: reserva de passagens de avião) (Gray, 1978). Uma transação é um conjunto de operações que têm as seguintes propriedades:

1) Atomicidade: todas as operações da transação são terminadas ou nenhuma é inicializada.

2) Sequencialidade: se várias transações são executadas em paralelo, seu resultado é o mesmo que se elas fossem executadas sequencialmente numa determinada ordem.

3) Permanência: quando uma transação é terminada, seu resultado é mantido ("commit" da transação).

Uma transação pode ela mesmo conter várias transações (transações aninhadas). Para permitir a atomicidade, o estado inicial da transação deve ser salvo e mantido durante toda a execução da transação para poder ser restituído:

- em caso de deteção de erro (pontos de recuperação);
- em caso de interbloqueamento ("dead--lock").

A salvação-restauração do estado inicial é facilitada pela utilização de "memória-estável" em disco (por exemplo, Tandem) ou em "memória rápida" (projetos ENCHERES (Banatre et atlii, 1986a) e GOTHIC (Banatre et alii, 1986b)).

### 3.2.3. Atores (em CHORUS (Zimmermann et alii, 1984)) ou Módulos (em CONIC (Kramer et alii, 1983))

São sequências de código ativadas pela chegada de mensagens e gerando outras mensagens. As comunicações entre "atores" (ou módulos) se expressam da mesma forma, estejam eles na mesma estação ou em estações diferentes. Para tolerar as faltas, "reconfigurações" permitem reorientar as mensagens destinadas a um ator sobre uma estação em falha em direção de um "ator" equivalente de uma outra estação (Banino et alii, 1985a). Uma aplicação é então constituída de execuções sucessivas de um conjunto de "atores" comunicando por mensagens. O controle da aplicação pode ser facilitado pela utilização de "mensagens de atividade" (Banino et alii, 1985b).

### 3.2.4. Modelo de Aplicação Orientado-Objeto

Neste modelo, toda entidade do sistema é um objeto. Um objeto, designado por um nome, possui um "estado" que pode ser consultado ou modificado por funções de acesso. O conjunto de objetos que têm as mesmas funções de acesso constitui uma classe. Por exemplo, uma palavra-memória, uma pilha, um arquivo, um processo são objetos.

O modelo orientado-objeto é, muitas vezes, implementado na forma de "tipos abstratos": é a "programação orientada--objeto".

Vários projetos de Sistemas Distribuídos são baseados neste modelo:

- nos Estados Unidos: ARGUS (Liskov, 1985), EDEN (Black, 1985), EMERALD (Black et alii, 1986), CLOUDS (Le Blanc & Wilkers, 1985), etc.
- na Europa: SOMIW (Shapiro, 1986), DASE (DASE, 1986), COMANDOS (Balter, 1986), GUIDE, etc.

O modelo orientado-objeto é bem adaptado ao controle dos direitos de acesso.

A fronteira entre estes modelos é bastante difusa. Assim, a execução de uma operação sobre um objeto pode ser considerada como a execução de um serviço no modelo cliente-servidor. Por outro lado, em certos projetos, as operações sobre os objetos são "atômicas", o que permite construir facilmente transações. Da mesma forma, os "atores" podem ser considerados como gerentes de objetos.

Nenhum destes modelos de aplicação distribuída é especializado para uma classe de aplicações dada, mesmo se alguns são melhor adaptados a determinadas classes (por exemplo, transações para os sistemas de bases de dados distribuídas). No entanto, nenhum destes modelos é reconhecido como suficientemente geral para ser adotado como modelo de referência para servir de base à construção de aplicações distribuídas integradas (no sentido dado no item I). Contudo, esta é a ambição do projeto ANSA (ANSA, 1987). Paralelamente, a Comissão das Comunidades Européias organiza um grupo de trabalho, dirigido por Hubert Zimmermann, encarregado de definir um "modelo de referência do processamento distribuído" (Distributed Processing Reference Model: DPRM), que deveria ser, ao nível das aplicações, o equivalente do OSI-RM ao nível das comunicações.

## 3.3. Sistemas Operacionais Distribuídos

Os sistemas operacionais distribuídos visam fornecer ao projetista de aplicação mecanismos que lhe permitam a realização eficaz dos modelos de aplicação. Estes mecanismos podem ser mais ou menos evoluídos.

O primeiro (também o mais antigo e o mais simples) destes mecanismos é a transferência de mensagens entre os processos de aplicação utilizando primitivas tais como SEND e RECEIVE, síncronas ou não, com ou sem difusão. De fato, trata-se de um mecanismo de base sobre o qual pode-se construir todos os outros mecanismos. Numerosos sistemas fornecem um serviço deste nível: CHORUS (Zimmermann et alii, 1984), CONIC (Kramer et alii, 1983), Amoeba (Mullender, 1985), Accent (Fitzgerald & Rashid, 1986), V-Kernel (Cheriton, 1984), Mach-1 (Rashid, 1986), etc. Uma extensão destes mecanismos de passagem de mensagem é constituído pelos fluxos, generalização dos "pipes" do UNIX.

Um mecanismo mais evoluído (mesmo que este ponto seja asperamente discutido pelos partidários dos sistemas a passagem de mensagens) é fornecido pela chamada de procedimento remoto (RPC: "remote procedure call"): isto consiste em fornecer uma interface idêntica à chamada de procedimento clássico, seja a execução local ou remota. A execução do RPC é bloqueante (síncrona) até o retorno do procedimento. O esquema de execução é, consequentemente, simples, pois retoma o esquema clássico de execução sequencial. Alguns projetos tornam "atômica" a execução de procedimentos remotos, permitindo deste modo a construção simples de aplicações tolerantes a faltas (projetos: CONCORDIA do programa ESPRIT, GOTHIC (Banatre et alii, 1986b), ARGUS (Liskov, 1985)) ou a construção de transações. Outras extensões do RPC consistem em torná-lo "não-bloqueante" (RSR: "remote service request" do projeto DELTA-4 (Bonn et alii, 1986)) para aumentar o paralelismo, ou a permitir as comunicações de m chamantes a n chamados ("multi-functions" (Banatre et alii, 1986b)).

Outros sistemas operacionais distribuídos implementam diretamente os modelos orientados-objeto e as operações correspondentes da linguagem que os suporta (CLOUDS (Le Blanc & Wilkes, 1985), SOMIW (Shapiro, 1986)).

## 4. QUALIDADES EXIGIDAS PARA OS SISTEMAS OPERACIONAIS DISTRIBUÍDOS

### 4.1. Transparência da Distribuição

O programador de uma aplicação distribuída deseja geralmente não ter que gerar o endereçamento das entidades que manipula: ele designa essas entidades por um nome, cabendo ao sistema operacional o encargo de associar um endereço ao nome conhecido da aplicação. Isto não exclui que certas entidades (sensores, atuadores, periféricos, servidores especializados, etc.) sejam localizadas em estações pré-definidas. Mas isto permite que certas partes da aplicação possam ser executadas em estações não pré-definidas, ou ainda, migrar de uma estação a outra em curso de execução, seja após uma falha (reconfiguração), seja para melhorar o desempenho (repartição de carga) (Theimer et alii, 1985). Assim é possível chamar pelo mesmo nome um conjunto de recursos ou "pool de servidores", com o sistema operacional escolhendo neste pool o servidor menos carregado ou o mais próximo do cliente. Este é o caso nos projetos "Cambridge Distributed Computing System" (Needham & Hebert, 1982), Amoeba (Mullender, 1985), SATURNE (Deswarte et alii, 1986a).

Em certos casos, o sistema operacional fornecerá à aplicação uma interface idêntica à de um sistema centralizado. Isto pode ser obtido gerando uma memória virtual ou um espaço de endereçamento único sobre toda a rede (sistema Domain d'Apollo (Leach et alii, 1983)), ou um sistema de gerenciamento de arquivos (Newcastle Connection (Brownbridge et alii, 1982), NFS (Lyon et alii, 1984)), ou ainda, fornecendo um interface sistema geral (Locus (Popek & Walker, 1985)).

Devemos notar que, por razões de tolerância a faltas ou de desempenho (proximidade), certas entidades (certos arquivos, por exemplo, ou ainda, tabelas de configuração ou de endereçamento) são copiadas no sistema distribuído, sendo os diferentes exemplares designados pelo mesmo nome. Em consequência, é importante manter a coerência deste, apesar dos acessos paralelos aos vários exemplares e apesar da presença possível de faltas (Locus (Popek & Walker, 1985), Saturne (Deswarte et alii, 1986a)).

### 4.2. Segurança de Funcionamento

A Segurança de Funcionamento ("Dependability") pode definir-se como a propriedade de um sistema que permite

colocar uma confiança justificada no serviço oferecido (Laprie, 1986). Segurança de Funcionamento é um termo genérico que visa cobrir ao mesmo tempo:

- a confiabilidade ("reliability") ou medida da continuidade do serviço:
- a disponibilidade ("availability") ou probabilidade de fornecer o serviço num instante dado, levando em conta as alternâncias entre os estados "em serviço" e "fora de serviço";
- a segurança ("safety") que mede a resistência a faltas catastróficas.
- a segurança ("security") que mede a resistência a faltas intencionais (intrusões).

Os sistemas distribuídos colocam um triplo problema do ponto de vista da segurança de funcionamento:

1) as ligações físicas são capazes de gerar erros (ruídos, parasitas, etc.);
2) a multiplicidade de equipamentos aumenta a probabilidade de falha e de intrusão;
3) a falha de um elemento é capaz de propagar rapidamente erros em elementos sem falhas.

O primeiro ponto pode geralmente ser resolvido pelos protocolos de comunicação (códigos detetores ou corretores de erro, repetição de mensagens, roteamento adaptativo) ou por dispositivos de comunicação redundantes (barramento duplicado, anel duplicado, etc.)

Os dois outros pontos necessitam a implementação de técnicas de impedimento a faltas ou de tolerância a faltas no conjunto do sistema distribuído, hardware e software.

O impedimento às faltas consiste em conceber, realizar e operar o sistema de forma a minimizar a probabilidade de ocorrência de faltas:

- qualidade da concepção do hardware e do software (métodos e ferramentas de especificação e de concepção);
- qualidade da realização (fabricação, componentes, integração, verificação);
- qualidade do ambiente de operação (alimentação elétrica, temperatura, hidrometria, formação do pessoal de operação, etc.).

As técnicas de impedimento serão desenvolvidas no item 4. Mas, qualquer que seja a "qualidade" do sistema, e portanto para tão baixa que seja a taxa de falha dos seus elementos, as consequências destas falhas poderiam ser de tal modo importantes que pode-se tornar indispensável a introdução de técnicas de tolerância a faltas.

A tolerância a faltas é obtida pela redundância, material e/ou temporal. Pode-se recorrer a duas técnicas para tolerar as faltas: a deteção dos erros e a recuperação destes ou o mascaramento dos erros.

A deteção e a recuperação dos erros comporta uma fase de deteção com o auxílio de mecanismos integrados ou acrescentados ao hardware e/ou software para efetuar o controle de verossimilhança sobre o estado do hardware, do software e dos dados: verificação das durações ("watchdog"), verificação dos valores (códigos detetores, deteção de código de instrução não-válido ou de endereço inexistente, controle de verossimilhança da aplicação, programas de testes, etc.), verificação dos direitos (de um processo efetuar certas operações sobre certos objetos).

Após deteção do erro, deve-se efetuar um diagnóstico dos elementos com falhas e, no caso da falta ser diagnosticada como permanente, operar uma reconfiguração para eliminar estes elementos e/ou acrescentar novos elementos permitindo continuar o tratamento. A seguir, é necessário corrigir o estado errôneo, seja por uma recuperação ("roll-back") sobre um estado anterior salvo ("backward recovery"), seja pelo estabelecimento de um novo estado correto, por exemplo, por reinicialização ("forward recovery").

O mascaramento de erros consiste em utilizar redundâncias, de maneira a poder continuar o tratamento mesmo em caso de erro: o estado do sistema é suficientemente redundante para permitir restabelecer um estado correto a partir do estado incorreto. O exemplo mais simples desta técnica é o voto majoritário: o tratamento é efetuado num número de exemplares suficiente para que os exemplares corretos sejam "majoritários"; os resultados dos diferentes

exemplares são votados e os resultados majoritários são considerados como corretos (Deswarte et alii, 1986a). Esta técnica é particularmente interessante para os sistemas em tempo-real, visto que os tratamentos (e então as durações de execução) são sempre os mesmos quer se tenha ou não erros. Devemos notar que a fronteira entre "mascaramento" e "deteção + recuperação" não está clara mas depende do ponto de vista do observador: um código corretor de erro corresponde ao mascaramento ou à deteção-recuperação?

Nos sistemas distribuídos, é muitas vezes tentador utilizar a redundância intrínseca do sistema para tolerar as faltas; a falha material de um elemento pode não bloquear o conjunto do sistema, e pode persistir um número suficiente de elementos corretos para continuar os tratamentos. Mas este princípio simples se choca com dois problemas:

- por um lado, o mascaramento necessita de uma redundância forte, geralmente mais importante que aquela que poderia ser fornecida "gratuitamente" pelos elementos inativos da rede (o projeto SATURNE (Deswarte et alii, 1986a) visa utilizar, da melhor forma possível, os elementos inativos para aumentar a redundância;
- por outro lado, a deteção-recuperação necessita de:
  * se precaver contra a propagação de erros: os mecanismos de deteção não têm geralmente uma cobertura de 1ØØ% e não atuam instantaneamente; os dispositivos de comunicação entre processos são, por conseguinte, susceptíveis de propagar erros e de contaminar outros processos, etc ;
  * definir "estados" do sistema, seja para salvar um estado em vista de uma recuperação ("backward recovery") seja para estabelecer um novo estado sadio ("forward recovery"); ora, é muito difícil observar um estado estável de um sistema distribuído, por causa do paralelismo elevado e da concorrência-cooperação entre processos; além disso, é desejável realizar somente o recobrimento dos processos contaminados e deixar continuar os outros. Utilizar "brutalmente" as técnicas de recuperação pode levar ao "efeito dominó" (Randell, 1975). Portanto, é indispensável conceber o software de aplicação em vista da recuperação e estruturá-lo em função dos mecanismos disponíveis: transações, conversação (Randell, 1975). Mas esta estruturação prejudica a transparência e impõe restrições que podem deteriorar o desempenho de forma significativa.

Por outro lado, existe uma classe de falhas "maliciosas" que devem geralmente ser levadas em conta na concepção de um sistema distribuído tolerante a faltas, pelo menos quando se deseja atingir um alto grau de segurança ("safety"): são as falhas "bizantinas" (do nome do problema dos generais bizantinos (Lamport et alii, 1982) Isto pode somente ser feito às custas de uma redundância mais forte ou de uma concepção específica (em particular dos meios de comunicação) (Wensley et alii, 1978).

As técnicas de tolerância a faltas aplicam-se evidentemente às falhas materiais, mas elas podem também ser adaptadas às falhas de concepção do software pela utilização da diversificação:

- deteção e recuperação: "recovery block" (Randell, 1975);
- mascaragem: programação em "N-versões" (Avizienis & Laprie, 1986).

A diversificação pode até mesmo atingir a especificação funcional.

Enfim, a segurança ("security") diante das falhas intencionais ("intrusões") deve ser particularmente estudada nos sistemas distribuídos por causa:

- da sua vulnerabilidade: é muito difícil proteger fisicamente um sistema geograficamente extenso;
- da gravidade potencial das sabotagens ou divulgações de informação.

É um dos pontos estudados dentro do projeto SATURNE (Deswarte et alii, 1986a).

## 5. CONCEPÇÃO, REALIZAÇÃO E OPERAÇÃO DE APLICAÇÕES DISTRIBUÍDAS

Em cada etapa do ciclo de vida de uma aplicação, convém prestar-se particular atenção

à qualidade de forma a evitar as falhas de concepção e de interação. Este problema é mais difícil para as aplicações distribuídas que para as aplicações sequenciais, pois o ser humano tem dificuldade em conceber e assimilar ações múltiplas que se desenvolvem em paralelo. As aplicações distribuídas apresentam maior complexidade e, consequentemente, maior possibilidade de erros.

## 5.1. Especificações funcionais

Partindo das especificações dos requisitos ("requirement specifications"), as especificações funcionais são a primeira etapa de concepção da aplicação distribuída. É uma etapa particularmente importante, pois uma falha de concepção, a este nível, terá consequências graves sobre as etapas seguintes e será tanto mais custosa de corrigir. Portanto, é conveniente verificar tanto quanto possível estas especificações funcionais segundo dois tipos de critérios:

1) Adequação aos requisitos: é necessário verificar que as especificações funcionais propostas satisfazem as exigências das especificações dos requisitos:
   - sob o aspecto funcional (a verificação pode ser manual);
   - sob o aspecto do desempenho: precisão, desempenho temporal,... (pode ser necessário o uso e simuladores);
   - sob o aspecto da segurança de funcionamento (por cálculos prévios de confiabilidade, disponibilidade, etc, eventualmente utilizando ferramentas tais como programa SURF desenvolvido no LAAS).

2) Coerência das especificações funcionais: deve-se verificar que as especificações sejam implementáveis e que a implementação possa funcionar (ausência de inter-bloqueios, de saturação,...).

Estas duas classes de verificações serão tanto mais fáceis e válidas quanto as especificações sejam escritas formalmente. Existem numerosos métodos de especificação formal; entretanto citaremos aqui apenas três, particularmente adaptados aos sistemas distribuídos:

1) As Redes de Petri: é uma representação gráfica de sistemas paralelos, mais sintética que as máquinas de estados finitos (ou autômatas), mas com a mesma potência de representação. Existem várias extensões das redes de Petri de base (redes de Petri "coloridas", "temporizadas", "estocásticas",...) melhor adaptadas a certas classes de problemas ou de verificação. Para verificar as redes de Petri podem ser utilizados dois tipos de ferramentas:
   - as ferramentas analíticas que permitem verificar formalmente certas propriedades tais como a ausência de inter-bloqueio, a ausência de saturação, etc.
   - os simuladores ou interpretadores que permitem verificar certas características (temporais, por exemplo).

2) Estelle (Courtiat et alii, 1987): linguagem de programação de algoritmos distribuídos, em curso de normalização pela ISO, é baseada sobre autômatos comunicantes, mais no estilo de uma linguagem procedural de tipo Pascal. Esta linguagem serve de base a um método de desenvolvimento que apresenta ferramentas:
   - de apoio à programação (manipulação de descrições formais),
   - de verificação formal de algoritmos,
   - de ajuste e simulação,
   - de geração de código.

3) Lotos (ISO, 86): é outra linguagem em curso de normalização pela ISO para a descrição de protocolos, mas baseada em outro formalismo (CCS, lógica temporal).

Estas duas linguagens são estudadas no projeto SEDOS do programa ESPRIT. Elas podem servir à especificação funcional de algoritmos distribuídos e à sua verificação formal, como também podem servir diretamente à programação e à geração de código.

Mesmo que as especificações funcionais não sejam descritas formalmente, é possível utilizar ferramentas de software para verificar algumas das suas características. Assim o simulador SPHINX desenvolvido no contexto do projeto SCORE (Le Lann, 1987)

permite a avaliação de desempenho de arquiteturas distribuídas.

5.2. Realização de Aplicações Distribuídas

Existem "Ambientes de Produção de Software" que permitem implementar, de forma eficaz, os algoritmos descritos por especificações funcionais. Mas poucos deles levam em conta as particularidades dos sistemas distribuídos. Uma exceção é o projeto SEDOS citado precedentemente.

A fase de realização deve também ser verificada por três tipos de validação:

1) Testes funcionais devem ser aplicados para ajustar e verificar os módulos desenvolvidos e sua integração. O paralelismo e a impossibilidade de observar o estado completo do sistema tornam particularmente difíceis os testes "exaustivos". Entretanto, algumas vezes, baterias de testes podem ser geradas automaticamente por analisadores de especificação formal. Por outro lado, ferramentas de testes interativos podem ser desenvolvidas para realizar testes "representativos" ou testes "agressivos" (Deswarte et alii, 1986b).

2) Devem ser efetuadas Verificações de conformidade com as especificações funcionais. Se as especificações funcionais foram expressas formalmente e se a linguagem de programação for adequada, é possível imaginar o estabelecimento de "provas" de equivalência entre as duas formas de algoritmos (especificação formal e programa). A execução simbólica pode permitir a obtenção de tais provas. Entretanto, essas técnicas são, no momento, apenas aplicáveis a programas sequenciais simples. Resta a percorrer muito caminho antes que elas possam ser aplicadas de forma eficaz aos sistemas distribuídos assíncronos.

   Portanto, é necessário recorrer a métodos mais formais como as "revisões de programa" por equipes de revisores.

   Uma outra via para garantir esta equivalência consiste em transformar automaticamente as especificações formais em programa executável. Isto é possível com Estelle,por exemplo, ou ainda por execução (interpretação) direta das redes de Petri. A linguagem de especificação torna-se então uma "linguagem de muito alto nível" (VHLL - 'very high level language'). Este tipo de interpretação das especificações formais pode igualmente servir à deteção de erros por comparação entre a execução do programa real e da interpretação da especificação (Ayache et alii, 1982): é uma utilização da diversificação do software.

3) É necessário igualmente realizar medidas para verificar que o sistema satisfaz às exigências das especificações de requisitos:
   - medidas de desempenho, por simulação do ambiente (ex: simulação de tráfego); existem ferramentas gerais para este tipo de medidas de desempenho, como LYNX, desenvolvido no contexto do projeto SCORE (Le Lann, 1987);
   - medição de certos parâmetros da segurança de funcionamento (cobertura, tempo de latência...) e verificação do comportamento do sistema na presença de falhas; para tal, usam-se "injetores de falhas", tais como o sistema MESSALINE desenvolvido no LAAS.

5.3. Operação

Na fase de operação de uma aplicação distribuída devem ser previstos três tipos de ações:

1) Ações de monitoração: convém monitorar o comportamento da aplicação, seja para ajustar certos parâmetros de forma a melhorar o desempenho, seja para detetar a presença de falhas de concepção residuais e as diagnosticar. A implementação de observadores (Ayache et alii, 1982) facilita a monitoração das aplicações distribuídas ou dos protocolos.

2) Ações de manutenção: a deteção e correção de falhas residuais de concepção conduz ao desenvolvimento de novas versões da aplicação. Estas versões devem ser validadas como os programas iniciais (ver ítem IV.2) e, também, podem ser verificadas por testes de "não-regressão" para

assegurar que nenhuma função presente na versão precedente não esteja ausente na nova versão. É desejável, às vezes, fazer coabitar várias versões da aplicação no sistema real para terminar a validação das novas versões (Deswarte et alii, 1986b).

3) Evolução da aplicação: é possível que, no decorrer do uso, novas necessidades surjam ou que o ambiente da aplicação mude ou que o suporte material seja modificado ou ainda que novas aplicações precisem ser integradas e cooperar com a aplicação existente. Isto pode levar a modificações completas da aplicação, para as quais o conjunto do desenvolvimento deve ser retomado. Evidentemente, este trabalho de modificação é facilitado quando tenham sido utilizados métodos formais.

Por fim, é preciso ressaltar a importância fundamental dos operadores humanos (e do pessoal de manutenção) durante a fase de operação. Muitas vezes, estas pessoas são susceptíveis de cometer erros mais graves e sobretudo menos previsíveis que as falhas materiais (Deswarte et alii, 1986b). É necessário então formá-los com cuidado e facilitar seu trabalho por uma interface amigável com o sistema. Deve-se ressaltar que os operadores são responsáveis pela grande maioria das intrusões na transferência ilícita de fundos, divulgação de informações confidenciais ou sabotagem. Consequentemente, é primordial limitar e verificar seus direitos sobre o sistema.

## 6. PARTICULARIDADES DOS SISTEMAS DISTRIBUÍDOS EM AUTOMAÇÃO INDUSTRIAL

Os Sistemas Distribuídos utilizados em automação industrial implementam geralmente os mesmos meios de comunicação, os mesmos modelos de aplicação, as mesmas linguagens, as mesmas ferramentas de desenvolvimento que os outros sistemas distribuídos. Entretanto, pelo menos para os mais críticos, se diferenciam pelas suas exigências em termos de tempo real.

"Um sistema é chamado 'tempo real' se alguma ou o conjunto das entidades são obrigadas a respeitar datas físicas para o início e o fim de suas execuções e se o desrespeito destas datas constitui uma falha do sistema" (Le Lann, 1987).

Para respeitar estas datas físicas, os Sistema Distribuídos Tempo Real impoêm geralmente limites máximos aos tempos de acesso na rede de comunicação. Isto pode conduzir a escolhas de alguns sub-conjuntos de protocolos normalizados que privilegiam o tempo de resposta (por exemplo, MAP (GM, 1986)). Mas para aplicações que têm exigências de confiabilidade e de segurança ('safety') rigorosas, estes limites devem levar em conta as hipóteses de falhas (por exemplo, perda de ficha, reconfiguração,...) o que pode levar a conceber protocolos específicos. É o caso, por exemplo, do protocolo 8Ø2.3D desenvolvido para o projeto SCORE (Le Lann, 1987) que é compatível com a norma IEEE 8Ø2.3 e Ethernet, mas que garante tempo de acesso limitado.

A consideração de datas físicas, que tem que ser respeitadas, pode ser integrada no sistema operacional distribuído (por exemplo, MARS (Kopetz et alii, 1982)). Mas existe uma outra estrutura da aplicação que é suscetível de se desenvolver no contexto dos sistemas distribuídos tempo real: trata-se dos sistemas "reativos" e das linguagens correspondentes (CCS, LCS, FP2, Esterel,...) (Perrin, 1987).

Entretanto, na medida em que as aplicações tempo real são somente uma parte do sistema distribuído industrial, tendo por conseguinte que cooperar com outras aplicações (gestão, planificação, CAD,...), é desejável adotar um modelo único para todas estas aplicações e funções idênticas ao nível dos sistemas operacionais distribuídos.

## 7. CONCLUSÃO

A nossa intenção foi pintar um retrato da pesquisa atual a respeito dos Sistema Distribuídos e tirar as principais orientações que devem determinar sua evolução nos próximos anos. Mas, se é relativamente fácil olhar para o passado e apresentar um histórico do desenvolvimento da informática nesta área, é muito mais difícil pretender tirar do atual universo difuso da pesquisa em Sistemas Distribuídos as tendências que se solidificarão no futuro. Somente o porvir permitirá julgar a exatidão das nossas conjecturas.

Reconhecimento

Determinadas partes deste artigo e da bibliografia foram influenciad s por (Krakowiak, 1987) e (Balter, 1986).

## 8. REFERÊNCIAS

ANSA, (1987). ANSA: Advanced Networked Systems Architecture. "Building a new future for Information Systems". PR.26.Ø7, January 1987, Cambridge, UK.

Avizienis, A. & Laprie, J.C., (1986). "Dependable computing: from concepts to design diversity". Proceedings of the IEEE, vol.74, no.5, May 1986, pp.629-638.

Ayache, J.M. & Courtiat, J.P. & Diaz, M., (1982). "Self-checking software in distributed systems". Proc. of the 3rd IEEE Int. Conf. on Distributed Computing Systems, Miami, October 1982, pp.163-17Ø.

Balter, R., (1986). "Systemes distribues sur reseau local: Analyse et classification". Bull, DSG/CRG/OSI, decembre 1986.

Banatre, J.P. & Banatre, M. & Lapalme, G. & Ployette, F., (1986a). "The design and building of Enchere, a distributed electronic marketing system". Comm. ACM, vol. 29, 1, January 1986, pp.19-29.

Banatre, J.P. & Banatre, M. & Ployette, F., (1986b). "An overview of the Gothic distributed operating system". Research Report no.284, INRIA-IRISA, January 1986.

Banino, J.S. & Fabre, J.C. & Guillemont, M. & Morisset, G. & Rozier, M., (1985a). "Some Fault-Tolerance Aspects of the CHORUS Distributed System". Proc. of the 5th IEEE Int. Conf. on Distributed Computing Systems, Denver, May 1985, pp.43Ø-438.

Banino, J.S. & Morisset, G. & Rozier, M., (1985b). "Controlling distributed processing with CHORUS activity messages". Proc. 18th Hawai International Conference on System Science, January 1985.

Black, A.P., (1985). "Supporting distributed operations: experience with Eden". Proc. of the 1Øth ACM SIGOPS, Washington, December 1985.

Black, A. et alii, (1986). "Distribution and abstract types in Emerald". IEEE Trans. on Software Engineering, December 1986.

Bonn, G. & Martin, P. & Powell, D. & Seaton, D.T., (1986). "Delta-4: Overall system specification (Issue 1)". LAAS Report no.86.276, August 1986.

Brownbridge, D.R. & Marshall, L.F. & Randell, B., (1982). "The Newcastle Connection - or UNIXes of the World Unite !". Software Practice and Experience, vol.12, 12, December 1982, pp.1147-1162.

Cheriton, D.R., (1984). "The V-kernel: a software base for distributed systems". IEEE Software, vol.1, 2, 1984, pp.19-42.

CNMA, (1986). "CNMA: Communication Network for Manufacturing Applications". ESPRIT Project 955, Proc. of the ESPRIT Technical Week, 1986, North-Holland.

Courtiat, J.P. & Dembinski, P. & Groz, R. & Jard, C., (1987). "ESTELLE : un langage ISO pour les algorithmes distribues et les protocoles". T.S.I., vol.6, no.2, 1987, pp.89-1Ø2.

DASE, (1986). "Preliminary draft for TR Dase". ICL contribution to ECMA TC32-TG2, April 1986.

Deswarte, Y. & Fabre, J.C. & Laprie, J.C. & Powell, D., (1986a). "A saturation network to tolerate faults and intrusions". Proc. of the 5th IEEE Int. Symposium on Reliability in Distributed Software and Database Systems, Los Angeles, January 1986, pp.74-81.

Deswarte, Y. & Alami, K. & Tedaldi, O., (1986b). "Realization, Validation and Exploitation of a Fault-Tolerant Multiprocessor: ARMURE". Proc. of the 16th IEEE Int. Symposium on Fault-Tolerant Systems (FTCS-16), Vienna, July 1986.

Fitzgerald, R. & Rashid, R.F., (1986). "The integration of virtual memory management and interprocess communication in Accent". ACM Trans. on Computer Systems, vol.4, 2, May 1986, pp.147- 177.

General Motors, (1986). "MAP specifications versions 2.1A & 2.2". August 1986.

Gray, J.N., (1978). "Notes on database operating systems". Operating Systems, Lecture Notes in Computer Science no.6Ø, Springer-Verlag, 1978, pp.393-481.

ISO/TC97/SC21/WG16-1 DP88Ø7, (1986). "Lotos: a formal description technique". July 1986.

Kopetz, H. & Lohnert, F. & Merker, W. & Pauthner, G., (1982). "The architecture of MARS". Technical University of Berlin, MA 82/2, April 1982.

Krakowiak, S., (1987). "Les systemes d'exploitation repartis : evolution recente et tendances de la recherche". T.S.I., vol.6, no.2, 1987, pp.151-161.

Kramer, J. & Magee, J. & Sloman, M. & Lister, A., (1983). "CONIC: an integrated approach to distributed computer control systems". IEE Proc. vol.13Ø, Pt. E, no.1, January 1983, pp.1-1Ø.

Lamport, L. & Shostak, R. & Pease, M., (1982). "The Byzantine generals problem". ACM Trans. on Programming Languages and Systems, vol.4, no.3, July 1982, pp.382-4Ø1.

Laprie, J.C., (1986). "Dependability: A Unifying Concept for Reliable Computing and Fault-Tolerance". LAAS Report no.86.357, December 1986.

Leach, J.P. & Levine, P.H. & Douros, B.P. & Hamilton, J.A. & Nelson, D.L. & Stumpf, B.L., (1983). "The architecture of an integrated local network". IEEE Journal on Selected Areas in Communications, November1983, pp.842-856.

LeBlanc, R. & Wilkes, T., (1985). "Systems programming with objects and actions". Proc. of the 5th IEEE Int. Conf. on Distributed Computing Systems, Denver, May 1985.

Le Lann, G., (1987). "Le projet SCORE: les systemes informatiques repartis temps-reel". T.S.I., vol.6, no.2, 1987, pp.175-178.

Liskov, B.H., (1985). "The Argus language and system". Distributed Systems: methods and tools". Ed. M. Paul & H.J. Siegerteditors, Lecture Notes in Computer Science no.19Ø, Springer-Verlag, 1985.

Lyon, B. et alii, (1984). "Overview of the SUN Network File System". SUN Microsystems Inc, October 1984.

Mullender, (1985). "Principles of Distributed Operating Systems Design". PhD Thesis Report, Mathematisch Centrum, Vrije Universiteit Amsterdam, Netherlands, October 1985.

Needham, R.M. & Herbert, A.J., (1982). "The Cambridge Distributed Computing System". Addison-Wesley ICS series, 1982.

Perrin, G.R., (1987). "Programmation parallele: point de vue sur les langages et les methodes". T.S.I., vol.6, no.2, 1987, pp.1Ø3-113.

Popek, G. & Walker, B.J., (1985). "The Locus distributed system architecture". MIT Press, 1985.

Randell, B., (1975). "System structure for software fault-tolerance". IEEE Trans. on Software Engineering, vol.SE-1, 2, June 1975, pp.22Ø-232.

Rashid, R.F., (1986). "Threads of a new system". Unix Review, August 1986, pp.37.

Shapiro, M., (1986). "Structure and encapsulation in distributed systems: the proxyprinciple". Proc. of the 6th IEEE Int. Conf. on Distributed Computing, Boston, May 1986, pp.198-2Ø4.

Svoboda, L., (1984). "File servers for network-based distributed systems". ACM Computing Surveys, vol.16, 4, December 1984, pp.353-398.

Theimer, M. & Lantz, K. & Cheriton, D., (1985). "Preemptable remote execuction for the V-System". Proc. of the 10th ACM Symp. on Operating Systems Principles, SIGOPS Operating Systems Review, vol.19, 5, December 1985, pp.2-12.

Wensley, J.H. & Lamport, L. & Goldberg, J. & Green, M.W. & Levitt, K.N.& Melliar-Smith, P.M. & Shostack, R.E. & Weinstock, C.B.,(1978). "SIFT: the design and analysis of a fault-tolerant computer for aircraft control". Proc. of the IEEE, vol.66, no.10, October 1978, pp.1255-1268.

Zimmermann, H., (1980). "OSI reference model". IEEE Trans. on Communications, April 1980.

Zimmermann, H. & Guillemont, M. & Morisset, G. & Banino, J.S., (1984). "CHORUS: a communication and processing architecture for distributed systems". INRIA Report no.328, September 1984.