SBA: Controle & Automação, Vol. 1, Nº 4, pp. 337-344

AMBIENTE DE PRODUÇÃO DE SOFTWARE PARA SISTEMAS DISTRIBUÍDOS EM TEMPO REAL

Joni da Silva Fraga Jean-Marie Farines

Laboratório de Controle e Microinformática
Depto. de Engenharia Elétrica - UFSC
88049 - Florianópolis - SC

Resumo:

Neste trabalho, são apresentados aspectos gerais de um ambiente de produção de software para sistemas informáticos distribuídos com aplicações em tempo real. A estratégia normalmente adotada para a produção de software é baseada no modelo ciclo de vida. Neste sentido, são discutidas as necessidades de ferramentas e métodos associados às diversas fases deste modelo, visando a produção de software distribuído. A integração de ferramentas segundo uma metodologia pode levar a um sistema coerente e com possibilidades de automação da produção de software; para tanto é apresentada uma proposta de um ambiente orientado segundo uma metodologia.

Abstract:

This work presents general aspects of a software production environment for distributed systems in real-time applications. The normally adopted strategy for software production is based on the life cycle model. The requirements in termes of tools and methods associated to the various stages of this model are discussed. The integration of tools according to a methodology may lead to a coherent system with possibilities of the automation of software production. To pursue this goal, a methodology-based environment is proposed.

1. INTRODUÇÃO

1.1. Aspectos Gerais da Produção de Software

O custo de desenvolvimento e manutenção do software, tradicionalmente considerado como tendo a maior contribuição no custo total do sistema, continua elevado apesar dos esforços realizados no sentido da procura de metodologias e de ferramentas para minimizá-lo. A persistência desta situação é devida a fatores tais como: velhos hábitos na produção de software, a diversidade de metodologias, a falta de integração entre as metodologias empregadas, a fraca penetração de novas tecnologias de produção de software na indústria e a consequente deficiência na verificação da adequação destas a situações reais.

Estes problemas estão sendo enfrentados através da criação de Ambientes de Produção de Software, que integram um conjunto de ferramentas correspondentes às várias fases da produção do software (requisitos informais, especificação funcional, projeto, implementação, testes e manutenção) para formar um sistema coerente e automatizado, com acesso simples e "agradável" aos vários tipos de usuários.

Num primeiro tempo, um ambiente consistia na simples junção de ferramentas já existentes, correspondentes a metodologias muitas vezes independentes, levando a uma redundância de informações pouco relacionadas entre si. Entretanto, a busca de uma maior uniformidade entre as ferramentas pode levar à integração construída em torno de uma metodologia como nos sistemas de programação CONIC (Sloman86), SARA (Campos77), CEDAR (Swinehart86) ou de uma linguagem como no ambiente APSE ("Ada programming support environment") para a linguagem ADA.

1.2. Características de Sistemas Distribuídos em Tempo Real

Os sistemas distribuídos têm se mostrado como a solução mais adequada para os problemas de automação industrial e controle de processo. São inúmeras as vantagens, apontadas na literatura, destes sistemas em relação a outros mais clássicos: maior disponibilidade e confiabilidade de serviços, melhor desempenho e facilidades para a manutenção e crescimento incremental.

Sistemas em tempo real são então implementados em sistemas multi-microcomputadores apresentando características especiais: estações especializadas e fisicamente distribuídas, acompanhando a distribuição do processo controlado. As estações não se apresentam de forma completamente autônoma mas cooperam ativamente para a execução da aplicação. Nestes sistemas, o suporte de interconexão, o sistema operacional e a aplicação se apresentam integrados, formando os ditos Sistemas Embutidos.

Aplicações em tempo real necessitam que se cumpram com rigor requisitos de integridade e de tempo de resposta (Le Lann87). A garantia do cumprimento destes requisitos depende da escolha adequada de algoritmos, das arquiteturas para implementá-los e também das metodologias e ferramentas escolhidas para a produção de software. A flexibilidade é também um dos requisitos mais importantes para sistemas integrados de computadores utilizados em automação; ela está associada às necessidades de sistemas evolutivos e com facil adaptação a mudanças do meio onde atuam.

Nos itens subsequentes são apresentados aspectos gerais de um ambiente de produção de software para sistemas informáticos distribuídos com aplicações em tempo real, e ainda uma proposta de um ambiente orientado segundo uma metodologia.

2. METODOLOGIAS

As metodologias hoje conhecidas não cobrem a totalidade das fases de produção do software e, por outro lado, seguem uma orientação que é função do tipo de aplicação pretendida. Em sistemas distribuídos, a produção do software é dificultada pelas características restritivas apresentadas nestes sistemas: ambiente propício a bloqueios, atrasos e a dificuldade de se estabelecer estados e tempos globais. As metodologias para este tipo de sistemas não são universalmente aceitas, entretanto, pode-se destacar alguns princípios gerais que norteiam a produção do software e que são apresentados a seguir.

2.1. Fases do Ciclo de Vida do Software

A estratégia normalmente adotada para a produção do software está baseada no modelo do ciclo de vida que consiste no sequenciamento das fases de análise de requisitos e especificações do sistema, projeto, implementação, testes e manutenção (Yau86).

2.1.1. Análise de Requisitos e Especificação do Sistema

Nesta fase, é obtida uma especificação do sistema correspondente aos requisitos exigidos pelo usuário. E' importante que esta especificação seja validada. O objetivo visado é a tentativa de evitar erros já nesta fase da produção do software, pela formalização e validação das especificações do sistema. A consequência deste procedimento é uma substancial diminuição nos custos.

A formalização das especificações está fundamentada em propriedades e regras semânticas bem definidas. Uma decorrência desta rigidez nas definições é a possibilidade de automatizar a análise das especificações.

Dos modelos propostos para representar as especificações de sistemas distribuídos, nenhum modelo formal simples de ser analisado consegue expressar a totalidade das características deste tipo de sistema. Redes de Petri e Máquinas de Estado apresentam a vantagem da simplicidade, mas podem levar a uma explosão de estados em sistemas complexos. A lógica temporal tem uma boa potencialidade para representar este tipo de sistema, mas, por outro lado, apresenta uma grande complexidade no uso. As linguagens SDL, Estelle e Lotos são também ferramentas de especificação formal, com possibilidades para a descrição de aplicações distribuídas (Courtiat87).

As metodologias normalmente utilizadas nas especificações, baseiam-se em técnicas descendentes ("top-down") que consistem na decomposição do sistema em subsistemas ou ainda em técnicas do tipo ascendentes ("bottom-up") que consistem na composição do sistema total a partir da junção de subsistemas. As ferramentas associadas a estas metodologias dependem do tipo de aplicação. No primeiro grupo citado estão as ferramentas baseadas nos métodos SADT (Ross77), Mascot (Simpson79), Análise Estruturada (De Marco78), (Ward86), (Gomaa86) e no segundo estão as metodologias Tipo Abstrato de Dados (Liskov75), Ocultamento de Informações (Parnas72) e Projeto Orientado para Objeto (Booch86).

2.1.2. Projeto

Nesta fase, é obtida a descrição do sistema que resulta das especificações funcionais da fase anterior; esta descrição inclui a apresentação da estrutura global do sistema e o detalhamento interno dos componentes deste.

A estrutura global é resultante das metodologias escolhidas a nível da fase anterior. Na descrição interna dos componentes do sistema a metodologia adotada baseia-se sempre em regras de estruturação de programas. Fluxograma, o Diagrama de Nassi-Scheiderman e Pseudo-linguagem são as ferramentas normalmente utilizadas para o detalhamento interno.

No caso de sistemas complexos, a abordagem

clássica para o projeto do software é o princípio de decomposição modular apresentado em (De Remer 76) que, entre outras coisas, favorece a reutilização do software. Este princípio propõe a programação de um sistema a partir de uma programação em pequena escala ("programming in the small") que trata com a construção dos componentes do sistema (módulos) e de uma programação em larga escala ("programming in the large") que consiste na composição do sistema a partir de um conjunto de componentes. Nestas condições, uma "configuração" de um sistema distribuído abrange a definição e a atribuição dos componentes às estações do sistema e as ligações entres estes componentes.

As decisões a respeito da melhor decomposição do sistema dependem normalmente de critérios informais do projetista; a formalização destes critérios deve se constituir na base para possíveis ferramentas de inteligência artificial de apoio ao projetista.

2.1.3. Implementação

As ferramentas necessárias para esta fase do ciclo de vida são ferramentas de apoio para a programação, tais como editores orientados para a sintaxe, compiladores, controladores de versões, gerenciadores de biblioteca, etc.

Um ambiente de produção de software normalmente tem associado uma linguagem (ou linguagens) de programação. As linguagens são diferenciadas pelas suas relações com o **Suporte de Tempo de Execução**; este, também chamado de "Kernel", é o responsável pela implantação do ambiente multi-tarefas que executa a aplicação distribuída. Muitas linguagens de programação, chamadas de linguagens de implementação de sistemas, incluem parte do suporte de tempo de execução. Neste caso, primitivas do suporte de tempo de execução se apresentam disponíveis ao programador embutidas em construções da linguagem utilizada (ou projetada) para o ambiente. Exemplos notórios destas linguagens são ADA (Barnes80) e MODULA-2 (Wirth82). No caso de linguagens convencionais é o sistema operacional que suporta o ambiente multi-tarefas; um exemplo disto é o sistema CHORUS (Zimmermann84) que apresenta uma interface para a linguagem Pascal.

2.1.4. Testes

Os testes são de dois tipos: testes na estação de desenvolvimento e testes de aceitação.

A nível de estação de desenvolvimento, são realizados os testes de componentes (módulos) baseados na análise interna destes e testes de integração do sistema que consistem na análise da correção das interfaces entre componentes. Para facilitar esta fase, devem estar disponíveis na estação de desenvolvimento, ferramentas de análise, de depuração, de geração de dados para testes, etc.

Os testes de aceitação que verificam a conformidade do sistema às especificações definidas pelo usuário são realizados no sistema alvo atuando no meio real ou numa simulação deste.

2.1.5. Manutenção

Nesta fase é efetuado o aperfeiçoamento, a adaptação e a correção do software em funcionamento. Esta etapa utiliza técnicas e ferramentas descritas anteriormente. Técnicas de reutilização de componentes de software e ferramentas como controladores de versão também podem ser envolvidas na manutenção.

2.2. Técnicas associadas a diversas fases do modelo ciclo de vida

Reutilização

A reutilização é um dos fatores principais da redução do custo de produção de software, pois elimina a necessidade do desenvolvimento de todos os componentes; procura-se utilizar o máximo possível do software existente. A reutilização dos componentes de software pode se dar não somente a nível de código, mas também nas especificações e no projeto; para tanto, são necessárias a padronização dos componentes do software e operações de (Biggerstaff 87): procura de componentes (com técnicas de classificação (Prieto-Diaz 87) ou gerenciadores de bibliotecas baseados em sistemas de inteligência artificial (Ramamorthy 86)), verificação da necessidade de modificação dos componentes, modificação e composição de componentes.

Técnicas de Medições

É importante ter à disposição ferramentas que permitam analisar a qualidade do software produzido em todas as fases do ciclo de vida; estas medições permitem orientar as decisões para melhorar o software produzido. Existem várias técnicas para medições da complexidade, da estabilidade, da confiabilidade e da disponibilidade do software (Yau 86).

3. ORGANIZAÇÃO DO AMBIENTE DE PRODUÇÃO DE SOFTWARE

O ciclo de vida do software para sistemas distribuídos em tempo real impõe diferentes requisitos tanto em termos de suporte de software como sobre o hardware. As tendências observadas mostram necessidades:

- de uma estrutura básica para o gerenciamento e a integração dos recursos distribuídos na rede; esta estrutura trata basicamente com os aspectos de tempo de execução (execução da aplicação em tempo real);

- de ferramentas adequadas para a produção do softwre.

Usualmente, a produção do software se dá em Estações de Desenvolvimento. Nestas máquinas está concentrado um conjunto de ferramentas que atuam nas diferentes fases do ciclo de vida do software. O software desenvolvido e testado é transferido às Estações de Execução para a operação do sistema total. As necessidades em termos de suporte (sistema operacional) e de ferramentas são diferentes para estes dois tipos de estações, conforme é apresentado a seguir. A figura 1 sintetiza a organização de um ambiente de produção de software.

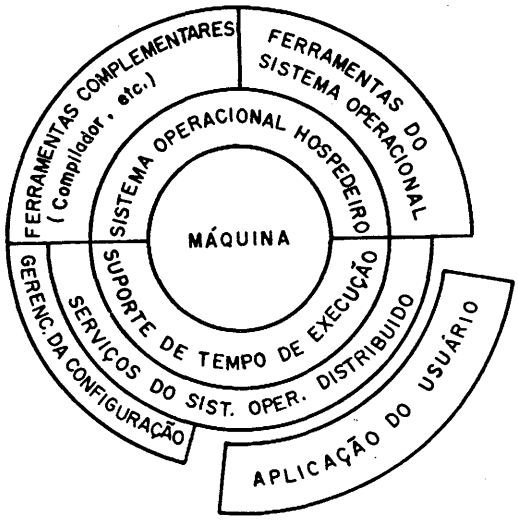

Fig. 1: Organização do Ambiente de Produção do Software

3.1. Máquinas e Suporte de Base

Devido à simplicidade das estações de execução, normalmente microcomputadores, equipamentos especializados (CLP, controladores de máquinas, etc), as estações de desenvolvimento devem apresentar recursos suficientes que permitam o bom funcionamento do sistema. É indispensável que estas estações apresentem unidades de disco com grande capacidade de memória, terminais gráficos, teclados, etc.

Sistemas de grande porte usualmente são desenvolvidos por um grupo de projetistas. Desta forma, o ambiente pode necessitar da produção distribuída com facilidades específicas para a integração das partes projetadas. Para isto, é necessário que ou a estação de desenvolvimento suporte vários terminais em "time sharing" ou ainda que o sistema apresente várias estações de desenvolvimento. Em ambos os casos são necessários mecanismos para que seja mantida a consistência das informações no(s) sistema(s) de arquivos e base(s) de dados mantidos pelo sistema.

Sistemas operacionais como Unix podem preencher grandemente as necessidades na produção de software. Exemplo disto são as ferramentas fornecidas pelo UNIX como o conector de programas Make e o controlador de versões SCCS.

3.2. Suporte de Tempo de Execução

Entre as funções a serem realizadas por um Kernel de sistema distribuído estão: comunicação e sincronismo entre tarefas, gerenciamento de tarefas, gerenciamento de memória e manipulação de interrupções e de entradas e saídas.

A arquitetura de um sistema distribuído está centrada sobre os mecanismos de comunicação e sincronismo ("inter-processus communications": IPC) (Watson81). Dois modelos básicos para a comunicação e sincronismo são identificados na literatura: chamada de procedimento e troca de mensagem. O modelo chamada de procedimento conserva as características de uma linguagem procedural com a dificuldade de manter em sistemas distribuídos a propriedade de que toda a chamada de procedimento deve sempre retornar (Nelson81). A comunicação e sincronismo para o modelo troca de mensagem se realizam por meio de duas construções, muitas vezes, não relacionadas sintaticamente ("send" e "receive"). Embora as construções no modelo troca de mensagens pareçam menos estruturadas se comparad s com o modelo chamada de procedimento, estas permitem maior flexibilidade em se tratando de sincronismo e possibilitam também um tratamento explícito de exceções. Estas características e a simplicidade fazem o modelo troca de mensagem mais apropriado para aplicações em tempo real e controle de processos.

Este ambiente de tempo de execução fornecido pelo "kernel" deve ter a comunicação e a sincronização uniformes e independentes da distribuição das tarefas no sistema. A extensão do IPC, de forma transparente sobre a rede de computadores é feita classicamente usando módulos ou tarefas "servidores de rede" (Rashid81), (Sloman86). Os requisitos colocados a um servidor de rede são: fornecer alguma forma de comunicação entre estações e acomodar as semânticas das primitivas de IPC, definidas localmente, quando de comunicação remota.

O suporte de tempo de execução não se limita só às estações de execução; é necessário que, principalmente as primitivas

de comunicação e sincronismo deste "kernel" sejam disponíveis na(s) estação(s) de desenvolvimento. Isto se deve ao fato que estas estações estão também envolvidas com informações fornecidas em tempo de execução (por exemplo, informações de configuração).

3.3. Serviços de Sistema Operacional Distribuído

A estrutura de serviços de OS nestes sistemas, é grandemente determinada pela interação entre as estações de desenvolvimento e de execução. Estas interações estão ligadas principalmente ao gerenciamento de configuração: transferência de arquivos de código, instanciação de componentes, monitoração do sistema, mudanças de configuração, etc.

3.4. Linguagem de Implementação de Sistemas

Seguindo o princípio de decomposição modular, a linguagem de implementação de sistema pode se apresentar composta de uma linguagem de componentes e uma de configuração.

3.4.1. Linguagem de Componentes

Os requisitos normalmente apontados para linguagens de programação em pequena escala são que inclua modularidade, verificação de tipos e compilação em separado. Classicamente módulos se apresentam divididos em interface que especifica um conjunto de operações ou estrutura de dados e corpo que corresponde à implementação das operações acessíveis através da interface. A separação de interface e corpo servem a vários propósitos:

- fornecer informação seletiva entre módulos implementadores e módulos clientes,
- apresentar interfaces de módulos compiláveis em tabelas de símbolos; estas tabelas são consultadas quando da ligação de módulos,
- restringir as relações de dependências entre dois módulos a importações: um módulo utiliza recursos implementados por outros módulos; e a exportações: o módulo implementa recursos utilizados por outros,
- viabilizar as importações de um módulo somente através de sua interface; isto permite separar a programação em pequena escala da programação em larga escala.

3.4.2. Linguagem de Configuração

As linguagens de configuração correspondem a uma evolução da clássica lista de comandos dirigida ao editor de "link", especificando os arquivos componentes a serem ligados. Estas linguagens agora apresentam construções de estruturação e verificação de tipos.

A composição de um sistema é exprimível em uma linguagem que permite definir o conjunto de tipos módulo a partir do qual o sistema será construído e operações tais como carga, instanciação e ligação de módulos. Estas operações devem ser executadas separadamente o que aumenta a flexibilidade na construção do sistema, permitindo carga e ligações incrementais (úteis para a configuração dinâmica). A configuração pode ser hierárquica, sendo composta de subconfigurações. Para tanto, são necessárias construções de estruturação na linguagem que permitam a construção de módulos a partir de módulos mais elementares.

A linguagem de configuração permite resolver as importações e exportações de maneira estática em relação à operação do sistema (configuração estática). A modificação de sistemas "on-line" devido a mudanças nas especificações de configuração (configuração dinâmica) é importante em aplicações de automação, onde não se pode interromper o processo controlado ou ainda diante de falhas parciais no sistema. Para permití-la é necessário prover construções de linguagem e facilidades de sistema operacional que permitam tirar e incluir módulos, durante a operação do sistema (Kramer86).

4. PROPOSTA DE UM AMBIENTE DE PRODUÇÃO DE SOFTWARE DISTRIBUÍDO PARA APLICAÇÕES EM TEMPO REAL

A produção eficiente de programas envolve o uso de metodologias e a existência de um ambiente integrado de ferramentas correspondentes às várias fases da produção do software. A finaldade dos ítens subsequentes é a de apresentar uma proposta de metodologia e ferramentas que viabilizem um sistema coerente e automatizado na produção de software.

4.1. Metodologia

A metodologia proposta segue as fases do modelo ciclo de vida. Alguns requisitos que podem ser colocados para a produção de software são: uma produção flexível e de baixo custo do software e a possibilidade de automação de grande parte do ciclo de vida. A produção do software segundo uma metodologia "orientada para o objeto" se mostra adequada aos requisitos citados pelas seguintes razões (Bruno 86), (Booch 86):

- fornecem modelos de software tidos como orientados para o problema; métodos mais clássicos produzem modelos orientados para a implementação;
- permitem a reutilização de componentes

do software, o que leva à redução dos custos.

Métodos funcionais de desenvolvimento não facilitam na maior parte das vezes as necessidades de mudança no espaço do problema, por serem métodos que apresentam critérios de decomposição onde os módulos resultantes são os mais abrangentes possíveis. As metodologias orientadas para objetos refletem uma melhor correspondência entre abstrações e os objetos do mundo real (Booch 86). As mudanças, neste caso, apresentam efeitos bem mais localizados do que em metodologias funcionais que dividem o sistema já no mais alto nível de especificação, estabelecendo com isto hierarquias e dependências de funções.

O conceito de herança ("inheritance"), que determina hierarquias de classes de objetos, é importante mas não necessário para caracterizar metodologias como orientadas para o objeto (Booch 86). A herança pode se traduzir na reutilização de especificações; uma classe é especializada numa subclasse pela modificação das especificações da primeira, adicionando novas características (Bruno 86). A reutilização, então, permite uma representação herdar propriedades de uma classe superior. Modelos de representação, tais como Rede de Petri, devem facilitar estas mudanças e permitir a rápida geração de código.

4.2. Modelo de Representação Formal

O modelo para a representação das especificações de componentes e do sistema global é Rede de Petri. O formalismo apresentado neste modelo é reconhecidamente apropriado para sistemas distribuídos. O modelo Rede de Petri preenche também os seguintes requisitos:

- simplicidade;
- representação formal das especificações que permita a validação prévia do comportamento do componente-objeto e do sistema;
- possibilidade de tradução automática das especificações;
- possibilidade de extensão do modelo (Rede de Petri temporizada, colorida, estocástica, etc.).

O modelo Rede de Petri apresenta a desvantagem da explosão de estados na formalização de sistemas complexos. Entretanto, a utilização de extensões da Rede de Petri simplifica a representação e a validação destes.

4.3. Proposta de Metodologia

A produção do software, segundo a metodologia proposta, distingue as fases de especificação/projeto, de implementação e de testes.

4.3.1. Especificação/Projeto

Esta fase está relacionada com a formalização e a validação das especificações do sistema.

A fase de especificação/projeto é composta respectivamente de uma etapa de descrição formal dos componentes-objetos e uma outra que trata da representação das especificações do sistema global.

a) Especificação dos Componentes-Objetos do Sistema

Esta etapa consiste na descrição da parte de controle do componente-objeto num primeiro passo e da parte de dados a seguir.

Primeiro Passo

O comportamento interno de cada objeto, do ponto de vista da estrutura de controle, é modelado por Rede de Petri; etiquetas descrevem as ações, as condições e a interface.

A atividade interna do objeto pode ser dividida entre tarefas (ou processos) dependendo do suporte de paralelismo fornecido pela linguagem de programação. Estas tarefas seriam identificadas pelas sequências cíclicas no modelo formal.

Segundo Passo

Neste passo, são definidas as estruturas de dados, tanto internas como as da interface do componente-objeto. As ações (procedimentos) e condições (predicados) associados aos lugares e transições do modelo Rede de Petri, representados no passo anterior por etiquetas, têm introduzidas as especificações de seus tipos. As especificações dos dados de interface são também introduzidas neste passo.

A representação dos dados deve apresentar compatibilidade com os tipos da linguagem de programação, de modo a garantir uma tradução automática.

b) Especificação do Sistema Global

A interconexão dos componentes-objetos, formando o sistema global, é também descrita no modelo Rede de Petri acrescido das especificações de dados. Informações de instanciação de objetos ou classes de objetos devem ser fornecidas neste momento; estas informações são importantes para tornar possível a tradução automática.

Na ligação entre componentes-objetos é feita verificação da compatibilidade dos

tipos de interface, a partir das especificações de dados.

c) Validação

A validação das representações formais são realizadas, primeiro individualmente por componente e posteriormente para o sistema global, usando técnicas automatizadas de análise para verificar a correção do modelo do ponto de vista da estrutura de controle e de simulação para estudar a evolução e o comportamento do sistema representado (controle e dados).

4.3.2. Implementação

A implementação consiste em traduzir na linguagem de programação as especificações. Esta tradução deve seguir os mesmos passos citados no ítem anterior.

A linguagem de implementação segue o princípio da decomposição modular que permite a decomposição desta em uma linguagem de componentes e uma linguagem de configuração. Estas linguagens devem apresentar as características citadas no item 3.4. A tradução automática das especificações em Rede de Petri está fortemente fundamentada na abordagem apresentada em (Bruno 86a) e (Bruno 86b):

Passo 1

Tradução das especificações de controle do objeto. Nesta tradução é importante considerar as informações sobre o objeto:

Informações estruturais derivadas da Rede de Petri

Estas informações estão ligadas ao número de tarefas, à sequência de suas operações e às suas comunicações com o mundo externo ao objeto. A tradução destas informações deve produzir um esqueleto do objeto que está associado somente aos aspectos de controle.

Informações sobre dados

Os tipos associados aos lugares de entrada e de saída da Rede de Petri, representando o componente-objeto e ainda as ações e predicados dos lugares e transições que compõem esta representação, são informações ainda não disponíveis que serão colocadas na forma de parâmetros formais nesta fase.

Passo 2

Neste passo, a tradução obtida do passo anterior é especializada com o acréscimo das traduções das especificações dos dados. Os tipos de portos (interface por mensagem), estados internos, funções de inicialização, ações e predicados estão definidos. As informações ligadas à importação (portos de saída) permanecem como parâmetros formais.

Passo 3

Este passo está ligado à tradução das especificações do sistema global na linguagem de configuração. O programa resultante apresenta declarações de contexto, das partes visíveis dos objetos (interfaces) e da conexão entre as mesmas.

4.4. Ferramentas

Uma das potencialidades oferecidas pelos modelos de representação formais está ligada à validação automática das especificações do problema. Neste caso, o ambiente de produção de software será provido, além das ferramentas usuais de edição, tradução, de ferramentas orientadas para a validação das especificações representadas por Rede de Petri.

As ferramentas de análise permitem verificar as propriedades gerais da rede de Petri (reinicialização, limitação, vivacidade, inexistência de bloqueamento, etc.) e as propriedades específicas ligadas ao comportamento desta (componentes conservativas, sequências cíclicas, etc.). Tais ferramentas são automatizadas e se fundamentam no uso combinado das técnicas de análise por enumeração de marcagens, redução e análise estrutural. Estas técnicas e ferramentas se limitam a Redes de Petri ordinárias.

A necessidade de conhecer melhor o comportamento do sistema modelizado por Rede de Petri e a sua evolução torna então indispensável o uso de ferramentas de simulação. Estas, apesar de não permitirem um estudo exaustivo do sistema, permitem validar modelos onde estão presentes temporizações, prioridades e influências do meio externo na forma de condições e ações e outras extensões (rede colorida).

As ferramentas de análise e simulação se completam dando informações a respeito da estrutura do modelo e da sua correção, no caso da análise, e do comportamento evolutivo deste, no caso da simulação. A presença destas dentro do ambiente de produção de software proposto é indispensável e deve permitir cobrir requisitos de validação de rede de Petri para vários tipos (ordinárias, temporizadas, coloridas, estocásticas, interpretadas, etc.).

5. CONCLUSÃO

O objetivo deste trabalho foi apresentar aspectos gerais sobre a produção do software para sistemas informáticos distribuídos. Foram abordadas metodologias, ferramentas e aspectos de organização para um ambiente de produção de software. Por fim, apresentamos uma proposta para aplicações em tempo real de um ambiente onde grande parte do processo de produção do software está integrado segundo uma metodologia.

6. BIBLIOGRAFIA:

Barnes, J.G.P. (1980). "An Overview of Ada", Software: Practice and Experience, Vol.10: 851-887.

Biggerstaff, T. & Richter, C. (1987). "Reusability Framework, Assessment and Directions", IEEE Software, Vol.4, No.2: 41-49, Mar 1987.

Booch, G. (1986). "Object-Oriented Development", IEEE Transactions on Software Engineering, Vol.SE-12, No.2: 211-221, Feb 1986.

Bruno, G. & Balsamo, A. (1986a). "Petri Net-Based Object-Oriented Modelling of Distributed Systems", OOPLSA'86 Conference: 284-293, Sep 1986.

Bruno, G. & Marchetto, G. (1986b). "Process-Translatable Petri Nets for The Rapid Prototyping of Process Control Systems", IEEE Software Engineering, Vol.SE-12, No.2: 346-357, Feb 1986.

Campos, I.M. & Estrin, G. (1977). "Concurrent Software System Design Supported by SARA at The Age of One", Proceedings of the Third International Conference on Software Engineering: 230-242.

Courtiat, J.P. & Dembinski, P. & Groz, R. & Jard, C. (1987). "Estelle: Un Langage ISO pour les Algorithmes Distribuès et les Protocoles", Techniques et Science Informatiques TSI, Vol.6, No.2: 89-102.

De Marco, T. (1978). Structure Analysis and System Specification , Ed. Prentice-Hall.

De Remmer F. & Kron, H.H. (1976). "Programming - in - the - Large Versus Programming - in - the - Small", IEEE Transactions on Software Engineering, Vol.SE-12, No.2: 80-82, June 1976.

Gomaa, H. (1986). "Software Development of Real Time Systems", Communications of ACM, Vol.29, No.7: 657-668, Jul 1986.

Le Lann, G. (1987). "Le Projet Score: Les Systèmes Informatiques Repartis Temps Reèl", Technique et Science Informatiques TSI, Vol.6, No.2: 175-178.

Liskov, B.H. & Zilles, S.N. (1975). "Specification Technique for Data Abstractions", IEEE Transactions on Software Engineering, Vol.SE-1, No.1: 7-19, Mar 1975.

Nelson, B.J. (1981). Remote Procedure CALL , XEROX Palo Alto Research Center, CSL-81-9, May 1981.

Parnas, D.L. (1972). "On The Criterio to be used in Decomposing Systems into Modules", Communications of ACM, Vol.15, No.12: 1053-1058, Dec 1972.

Prieto-Diaz, R. & Freeman, P. (1987). "Classifying Software for Reusability", IEEE Software, Vol.4, No.1: 6-16, Jan 1987.

Ramamoorthy, C.V. & Garg, V. & Prakashi, A. (1986). "Programming in the Large", IEEE Transactions on Software Engineering, Vol. SE-12, No.7: 769-783, Jul 1986.

Rashid, R. & Robertson, G. (1981). "Accent: A Communication Oriented Network Operating System Kernel", ACM Sigots Proceedings of the 8th Symposium on Operating Systems Principales: 64-75, California, Dec 1987.

Ross, D.T. (1977). "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Transactions on Software Engineering, Vol.SE-3, No.1: 16-34, Jan 1977.

Swinehart, D.C. & Zellweger, P.T. & Beach, R.J. & Hagmann, R.B. (1986). "A Structural View of the Cedar Programming Enrinonment", ACM Transactions on Programming Languages and Systems, Vol.8, No.4: 419-4990, Oct 1986.

Ward, P.T. (1986). "The Transformation Schema: An Extension of the Data Flow Diagram to Repeat Control and Timming", IEEE Transactions on Software Engineering, Vol. SE-12, No.12: 196-210, Jul 1986.

Watson, R. W. (1981). "Distributed Systems Architecture Model", Lectures Notes in Computer Science 105: 10-43, Ed. Springer-Verlag.

Wirth, N. (1982). Programming in Modula-2 , Ed. Springer-Verlag.

Yau, S.S. & Tsai, J. (1986). "A Survey of Software Design Techniques", IEEE Transactions on Software Engineering, Vol.SE-12, No.6: 713-721, Jun 1986.

Zimmermann, H. & Guillemont, M. & Mousset, G. & Banino J.S. (1984). "Chorus: A Communication and Processing Architecture for Distributed Systems", Rapport de Recherche INRIA No.328, Sep 1984.