ESTON ALMANÇA DOS SANTOS

VERIFICAÇÃO DE REQUISITOS DE SISTEMAS UTILIZANDO REDES DE PETRI

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

São Paulo
2002

OK

ESTON ALMANÇA DOS SANTOS

VERIFICAÇÃO DE REQUISITOS DE SISTEMAS UTILIZANDO REDES DE PETRI

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. José Reinaldo Silva

São Paulo
2002

Aos meus Pais, que têm sido a razão para o constante aperfeiçoamento de minha carreira. Que minha dívida para com eles seja eterna.

AGRADECIMENTOS

A Deus, pelo direito à vida e pelas oportunidades concedidas.

Ao meu orientador Prof. Dr. José Reinaldo Silva pela constante orientação e inestimável apoio.

Aos colaboradores e clientes da BCS Informática que, mesmo indiretamente, apoiaram esta iniciativa.

A todos que, de alguma forma, contribuíram para a realização deste trabalho.

RESUMO

A crescente demanda para a evolução do software, em função da disponibilidade de hardware mais avançado, gerou a necessidade da criação de técnicas e metodologias para suportarem esta investida. Desde então surgiram várias técnicas, da Análise Estruturada nos anos 70 às metodologias Orientadas a Objetos nos anos 90. Para a Engenharia de Software tornou-se extremamente importante a fusão destas técnicas, primeiro pelo ganho das metodologias resultantes e segundo pela introdução da padronização dos modelos destes sistemas. Como resultado das fusões de diversos métodos surgiu o Processo Unificado de Desenvolvimento de Software, que em paralelo definiu a UML como sendo uma linguagem de modelagem para especificação, construção e documentação de sistemas. Essa linguagem permitiu que a tarefa de análise e eliciação de requisitos se tornasse mais disciplinada, além de uma aproximação maior com o usuário, que é quem melhor entende do processo de negócio do sistema em desenvolvimento. Em função da informalidade desta fase, tornou-se mais difícil o processo de validação dos requisitos do sistema, uma vez que erros de modelagem nas fases preliminares identificados mais adiante têm um custo muito elevado, proporcionalmente ao tamanho e à complexidade do sistema. Esta dificuldade se agrava ainda mais quando consideramos os Sistemas de Informação para a Automação, que demandam uma integração vertical entre os processos gerenciais e as atividades de chão de fábrica. Com o objetivo de tornar possível a minimização do número de erros na fase de requisitos, neste trabalho será apresentada uma proposta que transforma os requisitos em especificação formal. Este processo é apresentado como sendo um refinamento da descrição disciplinada dos Use Cases, utilizando Redes de Petri como linguagem formal e ferramenta de validação.

ABSTRACT

The availability of hardware facilities for computer systems has increased the demand for large software and generated the need for new techniques and methodologies to support this offensive. Several paradigms have appeared, from the Structured Analysis in the 70's to Objects Orientation in the 90's. The fusion of these techniques are extremely important for Software Engineering, first because of the increased potential of the resulting methodologies, and second by the introduction of standard models for the target systems. The Unified Software Development Process is the most accepted result of the fusion method concerning object-oriented approaches, which have defined the UML as a language of modeling for specification, construction and documentation of systems. The language also allows that Requirements Elicitation and Analysis could be disciplined always centered in the user who understands better the business process. Due to the informality of the first step in the life cycle, the validation of the system requirements became more difficult, leading to very costly errors, which are proportional to the size and complexity of the project. Such difficulty becomes even worst when we consider Information Systems for Automation as a target. These systems demand a vertical integration between the management processes and shop floor activities. The requirements phase's errors can be minimized by the conversion in formal specification of the systems requirements, which is the proposal of this work. This process is presented as a Use Cases specification refinement, by using Petri net as formal language and validation tool.

PALAVRAS CHAVES

Engenharia de Requisitos, Validação de Requisitos, Redes de Petri, Processo Unificado de Desenvolvimento de Software, UML.

Errata

Legenda do "Tipo" da revisão:

A - Alteração
I - Inclusão
E - Exclusão

PÁGINA	LINHA	TIPO	ONDE SE LÊ	LEIA-SE
v	9, 10	A	em paralelo definiu	utiliza
v	11	A	sistemas.	seus modelos.
vi	22	A	validation	verification
viii	7	A	1.1 Motivação 2	1.1 Métodos pesquisados 2
viii	8	A	1.2 Métodos e técnicas utilizadas 2	1.2 Organização do trabalho 4
viii	9	E	1.3 Organização do trabalho 4	
viii	18	A	2.2.2 Dirigido a Use Cases 20	2.2.2 Dirigido a Use Cases 19
ix	18	A	3.3.2 Segmento da rede de Petri equivalente à descrição do Use Case 49	3.3.2 Segmento da rede de Petri equivalente à descrição do Use Case 48
ix	19	A	3.3.3 Rede de Petri equivalente à descrição proposta do Use Case 53	3.3.3 Rede de Petri equivalente à descrição proposta do Use Case 52
ix	20	A	4 RESULTADOS 55	4 RESULTADOS 54
ix	21	A	4.1 Especificação da Ferramenta de Verificação de Requisitos 55	4.1 Especificação da Ferramenta de Verificação de Requisitos 54
ix	22	A	5 DISCUSSÃO 58	5 DISCUSSÃO 57
x	1	A	5.1 Objetivos Atingidos 58	5.1 Objetivos Atingidos 57
x	2	A	5.2 Limitações da Proposta 59	5.2 Limitações da Proposta 58
x	3	A	5.3 Tendências das Ferramentas de Suporte a Engenharia de Requisitos 59	5.3 Tendências das Ferramentas de Suporte a Engenharia de Requisitos 58
x	4	A	6 CONCLUSÃO 62	6 CONCLUSÃO 61
x	5	A	6.1 Trabalhos Futuros 62	6.1 Trabalhos Futuros 61
x	6	A	LISTA DE REFERÊNCIAS 64	LISTA DE REFERÊNCIAS 63
xi	12	A	Figura 3-4 Rede de Petri equivalente ao Use Case Saque de Dinheiro 54	Figura 3-4 Rede de Petri equivalente ao Use Case Saque de Dinheiro 53
xi	13	A	Figura 4-1 Arquitetura da ferramenta PN-RAVaT 57	Figura 4-1 Arquitetura da ferramenta PN-RAVaT 56
xii	4	A	Tabela 5-1 Ferramentas de apoio a Engenharia de Requisitos 60	Tabela 5-1 Ferramentas de apoio a Engenharia de Requisitos 59
2	1	E	1.1 Motivação	
2	2	A	O autor se sentiu motivado em realizar o presente trabalho quando identificou o problema anteriormente apresentado e decidiu pesquisar alternativas que pudessem contribuir para a minimização das falhas apresentadas nos processos de desenvolvimento de sistemas.	A motivação em realizar o presente trabalho veio da identificação do problema anteriormente apresentado, o que incentivou a pesquisa de alternativas que pudessem contribuir para a minimização das falhas apresentadas nas fases iniciais, conforme descrevem os processos de desenvolvimento de sistemas.
2	6	A	São apresentadas a seguir a escolha dos métodos e técnicas utilizadas na pesquisa assim como a organização do trabalho.	São apresentados a seguir o estudo dos métodos utilizados na pesquisa assim como a organização do trabalho.
2	8	A	1.2 Métodos e técnicas utilizadas	1.1 Métodos pesquisados
3	11	A	o autor analisa	são analisados
3	16	A	o autor acredita	acredita-se
3	24	A	o autor propõe	propõe-se
4	8	A	desenvolvimento.	modelagem desses sistemas.

PÁGINA	LINHA	TIPO	ONDE SE LÊ	LEIA-SE
4	20	I		O objetivo do presente trabalho é somente propor este formalismo, baseado em redes de Petri e seguindo alguns conceitos do Processo Unificado. Será usada a UML representar parte da proposta (os diagramas e especificações de Use Cases) e as redes de Petri para a parte dinâmica que será a verificação dos Requisitos.
4	20	A	1.3 Organização do trabalho	1.2 Organização do trabalho
5	4	A	o autor apresenta	é apresentada
6	22	A	O autor concorda com	Do ponto de vista do autor
6	22	A	afirmação,	afirmação faz sentido,
12	20	A	não há no	o
12	21	A	nenhum compromisso com estes.	não deve ser entendido como sendo um método de desenvolvimento de software.
13	21	E	Isto mostra uma certa conveniência para os sistemas de informação, pois geralmente operam nesta arquitetura.	
16	11	A	o autor apresenta	apresenta-se
18	1	E	O objetivo do presente trabalho é somente propor este formalismo, baseado em redes de Petri e seguindo alguns conceitos do Processo Unificado. Será usada a UML representar parte da proposta (os diagramas e especificações de Use Cases) e as redes de Petri para a parte dinâmica que será a verificação dos Requisitos.	
21	9	I		(Rational Software Corporation, 2001)
25	3	A	$\cup (P \times T)$	$\cup (T \times P)$
25	4	A	IN	N
25	9	A	fluxo	fluxo.
27	1	A	A_i	M_i
27	4	A	$M^{'} = M + A_i \ \sigma$	$M_i = M_0 + A^T \sum_{0}^{i-1} \sigma_i$
27	6	A	M', é o estado posterior $\rightarrow M'$	M_i, é o estado posterior $\rightarrow M_i$
27	7	A	M, é o estado atual $\rightarrow M$	M_0, é o estado atual $\rightarrow M_0$
27	8	A	A_i,	A,
27	9	A	σ, é o vetor de disparo $\rightarrow \sigma$	σ_i, é o vetor de disparo $\rightarrow \sigma_i$
29	3, 4	A	da descrição do mesmo. Para tanto, o autor propõe	de sua descrição. Para tanto, propõe-se
30	14	A, E, I	O autor propõe que seja incluída, explicitamente, uma atividade que possibilite a "Verificar a consistência dos Requisitos do Usuário", que é a garantia de que eliciação	A inclusão, explicitamente, de uma atividade que possibilite a "Verificar a consistência dos Requisitos do Usuário", é a garantia de que a eliciação
31	4, 5	A	O autor acredita que esta	Esta
31	8	A	A seguir o autor justifica, com base no levantamento efetuado e na sua pequena experiência no desenvolvimento de software,	Justifica-se a seguir, com base no levantamento efetuado,
34	23	A	proposta pelo autor,	proposta,
36	8	A	O autor considera que a	A
39	23	I		3.2.3.1 Fluxo Principal de Eventos

PÁGINA	LINHA	TIPO	ONDE SE LÊ	LEIA-SE
40	20	A	3.2.3.1 Fluxo Alternativo de Eventos	3.2.3.2 Fluxo Alternativo de Eventos
46	18	A	corrente, pressupondo um sistema pré-emptivo.	corrente.
49	6	A	uma	um
49	10	A	P (Título - Descrição; (	p (Título - Descrição; (
49	11	A	P (Título - Descrição;	p (Título - Descrição;
50	1	A	P(Condição - Título - Descrição;	p(Condição - Título - Descrição;
52	1, 2	A	validação	verificação
55	2	A	o autor apresenta	é apresentada
56	5	A	O autor acredita	Acredita-se
58	2	A	O autor apresenta	É apresentada
59	7	E	pelo autor	
59	12, 14	A, E	idéia do autor e a falta de uma ferramenta que possibilite ao usuário visualizar interativamente o modelo em rede de Petri do respectivo Use Case	proposta deste trabalho e a falta de uma ferramenta que possibilite ao usuário visualizar interativamente o modelo em rede de Petri do Use Case
61	8, 10	A	o autor preferiu não avançar muito na proposta de um novo método ou da fusão mencionada entre os métodos citados sem antes conseguir juntar ao esforço inicial de análise e validação a	preferiu-se não avançar muito na proposta de um novo método ou da fusão mencionada entre os métodos citados sem antes conseguir juntar, ao esforço inicial de análise e validação, a
61	17, 18	A	todo o autor está sempre direcionando os esforços	todo, os esforços foram direcionados
62	9	A	O autor acredita	Acredita-se

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ATM	– Automated Teller Machine
BNF	– Backus Naur Form
CASE	– Computer Aided Software Engineering
CIMOSA	– Computer Integrated Manufacturing Open System Architecture
CORE	– Controlled Requirements Expression
DB	– Data Base
KB	– Knowledge Base
OMG	– Object Management Group
OMT	– Object Modeling Technique
OOSE	– Object Oriented Software Engineering
RUP	– Rational Unified Process
SRS	– Software Requirements Specification
SADT	– Structured Analysis and Design Technique
UML	– Unified Modeling Language
VORD	– Viewpoint Oriented Requirements Definition
VOSE	– Viewpoint Oriented System Engineering

LISTA DE SÍMBOLOS

●	Início de um processo (Use Case)
◉	Fim de um processo (Use Case)
→	Início de um evento do fluxo principal do processo
↳	Início de um evento do fluxo alternativo ao fluxo principal do processo
◇	Início de um evento condicional
$\rightsquigarrow^{n}$	Desvio da iteração corrente para a iteração ^{n}
||	Seqüência de eventos concorrentes

1 INTRODUÇÃO

A acentuada evolução do hardware propiciou a concepção de sistemas cada vez mais complexos. Em função disso, houve uma demanda para técnicas que suportassem o desenvolvimento desses sistemas, garantindo um nível de qualidade cada vez maior. Da Análise Estruturada, nos anos 70 e que na década de 80 evoluiu com a Análise Estruturada Moderna (Yourdon, 1989), até os métodos Orientados a Objetos que praticamente se consolidaram com o Processo Unificado de Desenvolvimento de Software (Jacobson, 1998), no final da década de 90, várias foram as contribuições de autores preocupados com a melhoria dos processos de desenvolvimento de sistemas. Por uma lado, a evolução dos métodos que suportam a análise e desenvolvimento de sistemas trouxe grande benefício para a comunidade que atua nesta área, através do surgimento de ferramentas de suporte à Engenharia de Software, as ferramentas CASE. Por outro lado, as atividades de Requerimentos de sistemas não gozaram do mesmo benefício.

O grande esforço no desenvolvimento de métodos e técnicas que focalizavam as fases posteriores à Engenharia de Requisitos gerou, basicamente, um problema. As falhas cometidas nas fases iniciais, muitas vezes pela falta de técnicas apuradas que pudessem identificá-las logo após a especificação dos Requisitos do sistema, freqüentemente só eram identificadas nas fases avançadas do ciclo de vida do desenvolvimento de sistemas.

1.1 Motivação

O autor se sentiu motivado em realizar o presente trabalho quando identificou o problema anteriormente apresentado e decidiu pesquisar alternativas que pudessem contribuir para a minimização das falhas apresentadas nos processos de desenvolvimento de sistemas.

São apresentadas a seguir a escolha dos métodos e técnicas utilizados na pesquisa assim como a organização do trabalho.

1.2 Métodos e técnicas utilizadas

A introdução do paradigma de objetos na Engenharia de Software alcançou a Engenharia em um processo de descentralização dos sistemas, no que se refere ao controle de automação de sub-sistemas associado a máquinas modernas, robôs, etc., porém com uma interrogação no que diz respeito a um formalismo para a fase inicial de modelagem e design de sistemas automatizados, a análise de requisitos.

Nos anos 90 surgem os sistemas de informação que denotam uma fase onde a automação explora mais o software (o supervisório, o sistema de apoio a decisão, o monitoramento em malha fechada, a integração vertical do chão de fábrica com os níveis gerenciais, etc) e portanto o impacto da abordagem para as etapas iniciais do processo de desenvolvimento, sendo assim a integração com os novos paradigmas de desenvolvimento de software, voltaram à ordem do dia.

Na área de Sistemas Automatizados, o ciclo evolutivo gerou, além da necessidade de se manter competitivo no mercado, uma demanda para que ocorresse a integração (vertical) entre os diversos sistemas disponíveis, desde os sistemas de chão de fábrica até aqueles que dão suporte à tomada de decisão (Vernadat, 1996). Portanto estes sistemas dependem muito do fluxo de informação no ambiente de fábrica, o que, em sistemas de produção automatizados circula com maior velocidade, além de depender da sincronia de vários processos ou sub-processos que funcionam de forma concorrente.

Portanto estes sistemas possuem um caráter dicotômico: por um lado a acurácia e a disponibilização da informação sobre o verdadeiro estado do chão de fábrica é essencial para o aumento de qualidade e flexibilidade no processo de tomada de decisão. Por outro lado, esta mesma característica depende do grau de integração (horizontal) do chão de fábrica e do controle que se tenha do fluxo de informação.

O desenvolvimento deste tipo de sistema demanda uma especificação concisa e muito acurada no que diz respeito ao comportamento do sistema. É portanto essencial que os métodos e a representação das especificações sejam tais, que seja possível expressar sem ambigüidades o comportamento do sistema e que seja possível verificar sua consistência.

Baseando-se na necessidade acima apresentada, o autor analisa os métodos baseados em Pontos de Vista do sistema, encontrados na Engenharia de Requisitos (Kotonya, 1998), além de explorar aqueles métodos que propõem a validação de requisitos como uma forma de garantir maior qualidade para os documentos de especificação de sistemas. Como uma evolução dos processos encontrados na Engenharia de Requisitos, o autor acredita que o Processo Unificado pode servir de base para se propor uma alteração na seqüência de ações no diagrama de Atividades do *workflow* de Requisitos deste processo.

A proposta de verificar as especificações do sistema se baseia na possibilidade de verificar a consistência lógica da descrição dos Use Cases do sistema, que são os diagramas utilizados no Processo Unificado para representar o fluxo de eventos de uma determinada funcionalidade do sistema. Este e outros diagramas fazem parte da Linguagem de Modelagem Unificada – UML (Unified Modeling Language). Para a consistência da descrição do fluxo de eventos do Use Case, o autor propõe representar estas especificações em Redes de Petri nas diversas fases do processo que leva do levantamento de requisitos até a aceitação de um conjunto de especificações sem ambigüidades, isto porque esta técnica permite realizar a verificação apoiada na sua representação formal dos seus modelos.

As redes de Petri surgiram na década de 60 com a tese de doutorado de Carl Adam Petri, com o objetivo de obter uma representação de sistemas concorrentes, e com o

passar do tempo ganhou força entre a comunidade acadêmica e industrial para a modelagem de sistemas de controle discreto e atualmente para modelagem de processos em geral.

Atualmente existem várias propostas de extensão ao que foi proposto inicialmente por Petri. No entanto, quando se compara os princípios de Engenharia de Software (Pressman, 1993) com as metodologias para o Desenvolvimento de Sistemas Dinâmicos, verifica-se que falta uniformidade e consistência no processo de desenvolvimento.

Neste trabalho não serão discutidas as potencialidades das redes de Petri, suas extensões e as redes de alto nível ou orientadas a objetos, para a representação de especificações, ou como linguagem formal ligada ao desenvolvimento de sistemas discretos. A proposta do trabalho se restringe a mostrar que, mesmo na sua forma elementar, as redes de Petri são uma representação bastante adequada (mesmo sem a introdução de recursos de abstração) para o processo de levantamento de requisitos às especificações de sistemas de informação.

Embora seja bastante tentador, ficará para um trabalho futuro a utilização de redes orientadas a objetos que poderiam ser uma representação transversal ao processo de desenvolvimento destes mesmos sistemas de informação, estando presente em todas as fases, desde as especificações até a representação de roteiros de teste.

1.3 Organização do trabalho

No próximo capítulo será apresentada a revisão da literatura que possibilitou apoiar a proposta deste trabalho. O capítulo 3 mostra a contribuição deste trabalho no sentido de permitir que especificações de requisitos sejam verificadas, enquanto que o capítulo 4 apresenta os principais resultados da aplicação da proposta. No capítulo 5 o autor discute sobre o alcance dos objetivos assim como aponta as limitações observadas durante a elaboração da proposta. Finalmente, no capítulo 6 é apresentada a conclusão, relacionando os possíveis trabalhos futuros.

2 REVISÃO DA LITERATURA

Com o objetivo de contribuir para o aumento do índice de detecção de erros durante a fase de levantamento de Requisitos, uma vez que os custos de correção de erros detectados nas fases posteriores se tornam muito elevados, o autor apresenta neste capítulo uma breve discussão sobre algumas das principais técnicas e processos atualmente desenvolvidos, que figuram entre as melhores práticas utilizadas pelos profissionais que atuam na área de desenvolvimento de sistemas. No próximo capítulo, será proposta uma técnica baseada em redes de Petri que permite que os requisitos sejam verificados, pelo menos quanto a sua consistência lógica.

Inicialmente, como revisão da literatura, será apresentado o processo de Engenharia de Requisitos (*Requirements Engineering)*, que disciplina a tarefa de levantamento e análise de requisitos, gerando o documento de especificação do sistema. Em seguida, será discutido o Processo Unificado *(Unified Process)*, que é um método recente para desenvolvimento de Software. Será abordada a *Unified Modeling Language* (UML), que é uma linguagem de modelagem para, dentre outros aspectos, especificar sistemas. Finalmente será apresentado o formalismo de redes de Petri, que se baseia na teoria de grafos, permitindo assim que processos integrados sejam formalmente modelados e que possuam ao mesmo tempo uma representação visual esquemática do comportamento dos mesmos e um mecanismo de verificação.

2.1 Engenharia de Requisitos

Neste trabalho, a Engenharia de Requisitos será vista como um processo sistemático e disciplinado de levantamento de necessidades e funcionalidades de um sistema. Portanto, a Engenharia de Requisitos transcende o artefato ou sistema em questão, e pode estar presente em outros processos de desenvolvimento particulares, como a Engenharia de Software.

Requisitos (*Requirements*) são definidos nas fases iniciais do desenvolvimento de sistemas como sendo uma especificação do que deve ser implementado. São descrições, geralmente em linguagem natural, de como o sistema deve proceder, ou de como as pessoas devem desempenhar determinado papel em uma organização.

Engenharia de Requisitos (*Requirements Engineering*) é o processo que engloba "todas as atividades envolvidas na descoberta, documentação e manutenção do conjunto de requisitos de um sistema baseado em computador" (Sommerville, 1997). Segundo Sommerville, o termo "Engenharia" significa que técnicas sistemáticas e repetitivas devem ser usadas de forma a garantir que os requisitos do sistema sejam completos, consistentes e relevantes.

A importância da aplicação deste processo pode ser observada na seguinte afirmação: "Diferentes pessoas geralmente executam processos de forma diferente em uma organização. Pessoas com mais experiência podem trocar a ordem das tarefas porque sabem das conseqüências do que estão fazendo. Por outro lado, os inexperientes necessitam seguir uma seqüência de execução das tarefas, pois não possuem este conhecimento" (Sommerville, 1997). O autor concorda com essa afirmação, pois durante sua experiência profissional presenciou situações em que, na falta de processos definidos para a execução de uma tarefa, pessoas com diferentes níveis de conhecimento executavam os mesmos processos de formas diferentes. Em alguns casos, a ausência de definições desses processos causavam até situações de desconforto dentro do ambiente de trabalho da organização.

Um dado importante é que, dependendo da complexidade do sistema a ser desenvolvido, deve-se considerar que o custo com as atividades de requisitos do sistema situa-se entre 10% e 15% (Kotonya, 1998) do custo total.

Considerando que os erros detectados em fases posteriores à fase de requisitos podem chegar a custar muitas vezes mais e que os problemas desta fase podem causar sérios desvios na fase de modelagem e design, como atraso na entrega do sistema, insatisfação do cliente, solicitações de alterações no sistema assim que o mesmo entra em produção, dentre outros inconvenientes, torna-se extremamente importante que as atividades desenvolvidas nesta fase possuam um reduzido índice de erros.

A seguir são apresentados os pontos básicos da Engenharia de Requisitos e as considerações mais importantes para o presente trabalho.

2.1.1 Documento de Requisitos

É um documento oficial dos requisitos do sistema que servirá como base de comunicação tanto entre clientes e gerentes de desenvolvimento, como entre usuários e desenvolvedores do sistema. Este documento é conhecido por vários nomes, dentre eles são citados alguns:

- Especificação Funcional (*Functional Specification*),
- Definição de Requisitos (*Requirements Definition*),
- Especificação de Requisitos de Software (*Software Requirements Specification – SRS*), etc.

Este documento pode possuir uma estrutura tão detalhada quanto se desejar possuir um padrão que garanta a qualidade dos documentos de requisitos dentro da organização. Uma estrutura mínima deve conter:

- Visão geral do sistema,
- Glossário,

- Lista de requisitos funcionais, e
- Restrições operacionais do sistema.

Por ser consensual, é importante destacar aqui a relevância do Documento de Requisitos, não apenas como registro do processo de levantamento e elaboração dos requisitos, mas como resultado, tanto parcial como final deste processo.

2.1.2 Processo de Engenharia de Requisitos

Pode ser entendido como um processo estruturado, composto de um conjunto de atividades que são seguidas para desenvolver, validar e manter um documento de especificação de sistemas. Um processo idealmente completo poderia incluir quais atividades seriam desenvolvidas, o encadeamento ou cronograma dessas atividades, quem é responsável por cada atividade, as entradas e saídas das atividades e as ferramentas utilizadas para suportar a Engenharia de Requisitos.

As entradas do processo de Engenharia de Requisitos são informações sobre os sistemas existentes, necessidades dos usuários, padrões da organização, informações de domínio do sistema, e regras reguladoras externas à organização. A aplicação deste processo varia entre as organizações, porém a maioria envolve eliciação de requisitos, análise e negociação de requisitos e validação de requisitos. A saída deste processo é o Documento de Requisitos do sistema.

Para representação deste processo utilizam-se modelos esquemáticos que são descrições simplificadas vistas de uma determinada perspectiva. Exemplos destes modelos são: modelos de atividades, modelos de funções (*role-action*) e modelos de entidade-relacionamento.

Fatores Humanos, Sociais e Organizacionais são influências importantes no processo de Engenharia de Requisitos. Eles geralmente estabelecem considerações técnicas.

A figura 2-1 ilustra o Processo de Engenharia de Requisitos, destacando as principais entradas e saídas do processo (Kotonya, 1998).

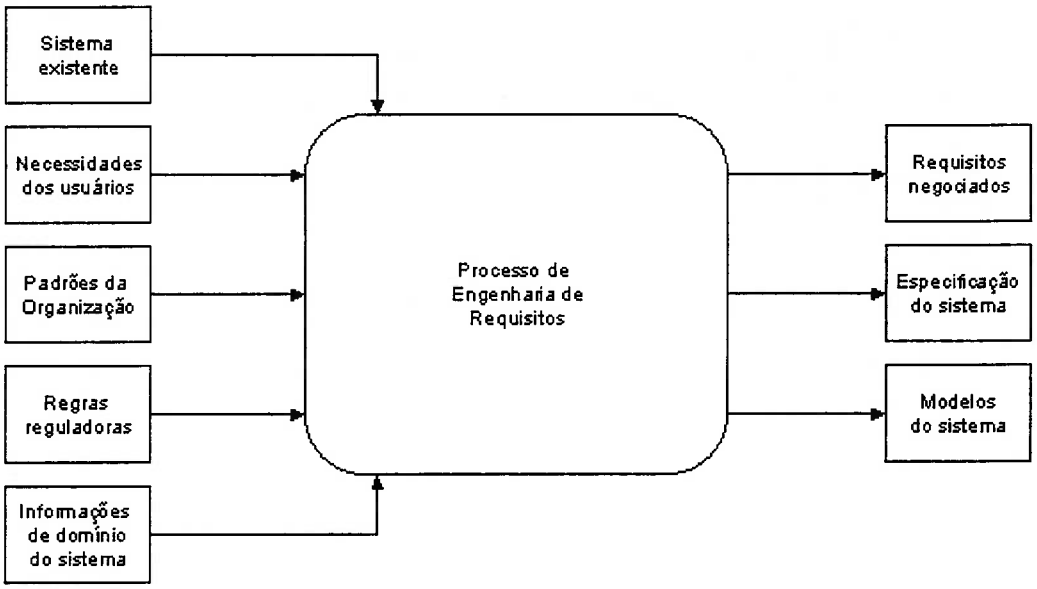


Figura 2-1 Entradas e saídas do processo de Engenharia de Requisitos

Da figura 2-1 pode-se destacar as entradas do processo de Engenharia de Requisitos que freqüentemente são desconsideradas quando não se executa esta tarefa: "Sistema existente" e "Regras reguladoras". Em muitos casos estas entradas são percebidas em fases adiantadas do processo de desenvolvimento e isto gera retrabalhos. As saídas deste processo são documentos formais que podem servir de base de negociação para contratação de equipes que farão a implementação ("especificação do sistema"), como também para acertos em casos de desentendimentos entre as partes ("requisitos negociados").

2.1.3 Validação de Requisitos

A validação dos requisitos se preocupa em checar os requerimentos quanto a omissão, conflitos e ambigüidades, e garantir que os requisitos sigam os padrões de qualidade definidos na organização.

De acordo Sommerville, o uso da prototipação poderia ser útil nesta fase, já que usuários poderiam utilizar o protótipo como elemento para validação do processo. Desta forma a fase de análise do sistema, mesmo sendo uma análise essencial, deve

ser antecipada, já que o protótipo deve conter funcionalidades do sistema, e não somente os requisitos, para a execução do processo de negócio do usuário.

Segundo (Sommerville, 1997), as principais tarefas a serem executadas durante o processo de validação de requisitos são:

- Certifique-se que o Documento de Requisitos atenda seus padrões
- Formalize a inspeção de Requisitos
- Utilize equipes multidisciplinares para revisar Requisitos
- Defina *check lists* de Validação
- Use protótipos para visualizar Requisitos
- Escreva um rascunho do manual do usuário
- Planeje Casos de testes de Requisitos
- Traduza os requisitos em modelos de sistemas

As abordagens em validação de requisitos buscam garantir que os documentos de especificação reflitam a necessidade do sistema em questão. Os casos informais, que não requerem especialização do analista e estão mais próximas dos usuários do sistema, tendem a ser mais frágeis, uma vez que basicamente se apóia em revisão de documentos e comparações com necessidades da organização. Por outro lado, aqueles casos que buscam tentativas de formalização tendem a ser mais confiáveis; no entanto requerem maior especialização dos analistas no sentido da necessidade de expressar os requisitos do sistema em uma determinada linguagem formal. Isto faz com que esta abordagem seja utilizada em situações bem específicas, por exemplo em aplicações que exijam um nível de erro bem reduzido.

2.1.4 Pontos de Vista (Viewpoints)

Para entender os requisitos de um sistema, deve-se entender os serviços que o sistema fornece, o domínio de aplicação do sistema, restrições não funcionais, o

processo de desenvolvimento do sistema, o ambiente onde o sistema será instalado e questões organizacionais que afetam a operação do sistema. Segundo (Kotonya, 1998), "o processo de Engenharia de Requisitos envolve a captura, análise e resolução de muitas idéias, perspectivas e relacionamentos de diferentes níveis de detalhes". No sentido de resolver este problema, o de estruturar os diversos conhecimentos de requisitos, alguns métodos têm surgido baseados na noção de "pontos de vista" (*viewpoints*).

O *Viewpoint* é "uma coleção de informações sobre um sistema ou um problema, ambiente ou domínio relacionado, que são apresentados de uma perspectiva em particular" (Kotonya, 1998). Estas perspectivas podem ser de usuários do sistema, de outros sistemas, de profissionais envolvidos no processo de desenvolvimento, etc. Geralmente, a informação de cada *viewpoint* é incompleta, porém os requisitos gerais do sistema são derivados da integração das informações de cada *viewpoint*. Portanto, é de se esperar que haja um processo de resolução de conflitos para eliminar as inconsistências entre as diferentes especificações originadas pelos diferentes *viewpoints*.

A seguir, de acordo (Kotonya, 1998), são relacionadas as principais técnicas que foram desenvolvidas para tratar os requisitos utilizando *viewpoints*.

2.1.4.1 Structured Analysis and Design Technique (SADT)

Esta técnica foi desenvolvida nos meados da década de 70 (Marca, 1988) e se baseia no modelo de fluxo de dados que visualiza o sistema como sendo um conjunto de atividades interativas. A SADT não possui um mecanismo explícito de *viewpoint*, entretanto eles podem ser vistos como extensões intuitivas desta técnica de modelagem já que cada módulo funcional é visto como um elemento que encapsula funções internas e interage por diferentes interfaces com o mundo externo (na maior parte dos casos composto por outros módulos, sistemas, etc.). Fontes de dados e repositórios constituem *viewpoints* nesta técnica. (Ross, 1985)

2.1.4.2 Controlled Requirements Expression (CORE)

Criado nos anos 70 para atender à indústria aeroespacial, o CORE tornou-se padrão na Europa e é até hoje usado pela "European Fighter Aircraft". É portanto um dos exemplos bem sucedidos de modelo de especificações utilizados tanto para sistemas (mecânicos, eletrônicos, etc.) quanto para software.

Fruto da linha de pensamento estruturada dos anos 70, o CORE também é baseado em decomposição funcional, como a SADT. Porém, ao contrário da SADT, nesta abordagem são observados, explicitamente, dois níveis de *viewpoints*: o primeiro nível é composto pelas entidades que interagem ou modificam o sistema (segundo o seu caráter ativo ou passivo estes *viewpoints* são denominados funcionais ou não-funcionais por seu autor); o segundo nível é composto pelas entidades que definem ou impõem limites ao sistema. Assim, neste nível as entidades que suportam os *viewpoints* são todas passivas (os *viewpoints* de fronteira são, em uma visão *top down*, entidades que interagem indiretamente com o sistema, colocando pela primeira vez a propriedade de se representar *viewpoints* indiretos). (Mullery, 1979)

2.1.4.3 Viewpoint-oriented System Engineering (VOSE)

O VOSE foi desenvolvido nos anos 90 no Imperial College por Finkelstein e Nuseibeh, (Finkelstein, et al., 1992) com o intuito de permitir a integração de métodos de desenvolvimento. Embora se interprete por "métodos de desenvolvimento" como sendo métodos de desenvolvimento de software, não há no VOSE nenhum compromisso com estes.

A filosofia básica do VOSE pode ser resumida no termo "*separation of concerns*", isto é, o fato de que no desenvolvimento de sistemas (mais até que no desenvolvimento de software) participam especialistas em vários aspectos do desenvolvimento de software e da área de aplicação. Portanto, é importante analisar como se modifica o ponto de vista de cada um destes agentes durante o processo de desenvolvimento.

No método VOSE, cada *viewpoint* padrão pode ter várias janelas, que seriam as sub-divisões dentro do próprio *viewpoint*. A figura 2-2 ilustra esta sub-divisão (Kotonya, 1998).

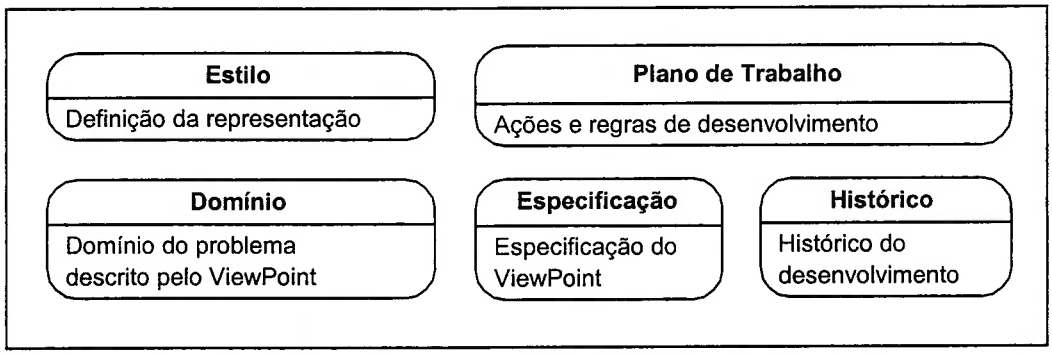


Figura 2-2 Janelas de um *Viewpoint* padrão

Associado ao *viewpoint* padrão, para cada agente pode ser associado um diagrama de fluxo de dados e um diagrama de estado (representando ações). A introdução de múltiplos diagramas na fase de requisitos é, portanto, apresentada explicitamente, pela primeira vez, na definição do VOSE.

A especificação de requisitos global do sistema é a combinação das visões de cada um dos agentes identificados, uma vez resolvidos os conflitos (Finkelstein, et al., 1992). A dificuldade para a utilização deste método é justamente a combinação das diversas visões (cujo número pode ser grande, principalmente em aplicações como sistemas de informação para manufatura). Como nos casos anteriores, não há nenhum método de validação previsto.

2.1.4.4 Viewpoint-oriented Requirements Definition (VORD)

O VORD foi inicialmente feito para sistemas interativos, onde existe uma classe de usuários e casos de uso na qual estes usuários passam informação para o sistema e, em troca, recebem serviços e/ou mais informações. O VORD funciona de forma análoga a um sistema cliente-servidor, onde os *viewpoints* são os clientes (Kotonya, 1996). Isto mostra uma certa conveniência para os sistemas de informação, pois geralmente operam nesta arquitetura.

Os requisitos são distribuídos em duas classes distintas:

- Diretos: que correspondem diretamente aos clientes, isto é, os que enviam informação e parâmetros e recebem de volta serviços;
- Indiretos: que correspondem aos *viewpoints* que não interagem diretamente e nem requerem serviços diretamente mas têm "interesse" nos serviços. São de fato *viewpoints* que não estão diretamente ligados a usuários do sistema e nem à forma como este interage com o mundo exterior, mas são muito importantes para o seu funcionamento adequado. Os aspectos organizacionais de sistemas de informação são exemplos claros deste tipo de *viewpoint*.

2.1.4.5 Viewpoint-oriented Requirements Validation (VORV)

O VORV (Leite, 1991) parte da constatação de que o processo de eliciação e, principalmente, de validação de requisitos em sistemas complexos tem se mostrado árduo, principalmente para sistemas de grande porte. Portanto é necessário que a validação seja inerente ao processo de levantamento de requisitos ao invés de encerrar o processo, após o levantamento de todos os requisitos, diretos e indiretos, relativos a diversas classes de usuários e sistemas.

O método é baseado em três conceitos básicos: um *viewpoint*, uma perspectiva e uma visão. O *viewpoint* é um estado mental de um indivíduo que examina o universo de discurso (o contexto em que o sistema está inserido). Uma perspectiva é a descrição de fatos e ações decorrentes de um *viewpoint* e de aspectos de modelagem. Finalmente, uma visão é a integração dos dois aspectos anteriores.

A aplicação do método pressupõe a utilização de *viewpoints* de pelo menos dois analistas – muito embora estes não sejam particularmente caracterizados. A seguir esses *viewpoints* são sobrepostos de maneira a integrarem uma perspectiva do sistema. Os conflitos entre os *viewpoints* são resolvidos antes de se fechar a perspectiva final do sistema. (Leite, 1991)

2.1.4.6 Uma discussão sobre os métodos de Requisitos de Sistemas

Os métodos de levantamento de Requisitos descritos anteriormente apontam para as características que as novas propostas devem conter. Embora não seja objetivo deste trabalho propor um novo método, mas sim apresentar uma contribuição parcial no sentido de formular um processo de verificação, é importante destacar estas características para fins de uma discussão mais ampla sobre o tema, que será apresentada no Capítulo 5.

As características a serem destacadas são:

1. os métodos apontam na direção da estruturação e hierarquização de agentes e *viewpoints*, como o CORE, VORD e VORV

2. a identificação dos agentes principais e seus respectivos *viewpoints* é de fundamental importância e é um elemento indispensável para integrar o levantamento de requisitos com o processo de geração de especificações

3. a classificação de *viewpoints*, seja conjuntamente com os agentes (VOSE) ou em separado (VORD, VORV), é fundamental para permitir a validação e para um bom entendimento dos próprios requisitos. Em geral, a classificação destaca os *viewpoints* "externos" e "diretos", envolvendo o núcleo do sistema e usuários, outros sistemas, etc. e os "internos", "indiretos", que não pressupõem interação ou troca de informação com o contexto, mas apenas o funcionamento interno do sistema

4. a validação, que não aparece explicitamente (ou apenas informalmente) nos demais métodos, é apresentada no VORV como resultante da estratégia de levantamento dos requisitos e integrada com esta. É opinião do autor que esta é a abordagem correta e que servirá de referência para futuras contribuições. A contextualização, que aparece no VORV, também é um aspecto de referência, permitindo a fundamentação dos *viewpoints* e portanto contribuindo para o processo de validação.

Além destas observações, seria importante introduzir um questionamento de natureza conceitual e prático: seria conveniente caracterizar um conjunto mínimo de classes

de *viewpoints* e agentes, cuja integração seria suficiente para justificar um novo projeto de sistema? Se existir tal classificação, esta deve estar de acordo com os métodos de desenvolvimento atualmente utilizados, como o Processo Unificado.

Embora, esclarecendo novamente, não seja o objetivo deste trabalho apresentar uma proposta (integral ou parcial) de método de levantamento de Requisitos, é importante, para contextualizar a proposta de verificação deste trabalho, mostrar que é possível integrar a validação no processo de eliciação dos Requisitos, e até avançar neste tópico, propondo uma alternativa à VWPL (*Viewpoint Language*) definida por (Leite, 1991), que pode ser um esquema formal genérico como as redes de Petri (ou até um sistema de redes hierárquica e/ou orientada a objetos).

A seguir, o autor apresenta uma possibilidade de especificar sistemas com um enfoque para a verificação dos Requisitos do mesmo.

Considerando três classes de atores hipotéticos, cujos *viewpoints* se deseja integrar como condição necessária para o sucesso de sistema de informação: o Usuário final, o Patrocinador do sistema, e o Analista (a figura 2-3 ilustra esta consideração). O Usuário final aparece em todos os métodos anteriormente descritos, embora não tenha sido destacado especialmente. O Patrocinador é o contratante e beneficiário do sistema, embora não seja necessariamente o seu usuário final. Este deseja que o sistema atenda aos seus interesses, satisfazendo o Usuário final.

O Analista é o que irá fazer o projeto e eventualmente a implementação do sistema, e portanto deve saber exatamente o que deve ser feito e quais as funcionalidades que devem ser atendidas.

Portanto a validação é uma integração harmônica destes *viewpoints*, como preconizado no VORV.

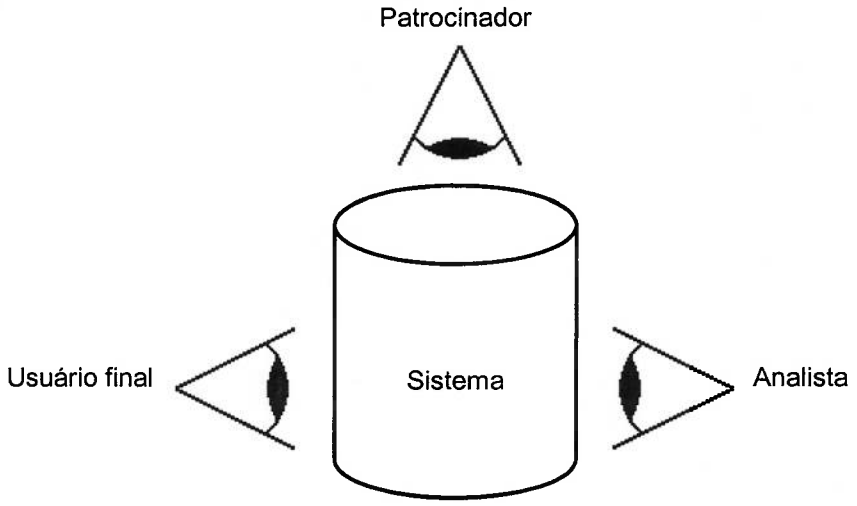


Figura 2-3 Pontos de vistas de atores de um Sistema

Estes são atores clássicos nos sistemas de informação, sejam estes destinados a serviço, negócios (*e-business*) ou para produção (*e-manufacturing*). Sob o ponto de vista do autor, um "bom sistema" seria, metaforicamente, um sistema cujos requisitos englobassem estes *viewpoints*.

Prosseguindo com a metáfora, destaca-se em primeiro lugar que, por ser a Engenharia de Requisitos baseada em eliciação e em processos intrinsecamente informais, esta admite uma dependência dos atores principais aqui citados. Segundo, que os requisitos eliciados, segundo o *viewpoint* de cada um destes atores, seria suficiente para determinar "fronteiras" do sistema (limites de separação entre o que é e o que não é o sistema), porém seriam insuficientes para encapsular todas as funcionalidades (seja por omissão de algumas, seja por falta de profundidade de visão sobre o sistema). Terceiro, o que poderia ser qualificado como um "bom sistema" (pensando, principalmente, nos sistemas de informação) seria representado por um conjunto de requisitos encapsulados pelas fronteiras de, pelo menos, estes três *viewpoints*.

Para que isto possa ser minimamente materializado seria necessário ter uma representação (preferencialmente formal) que integrasse todos os *Use Cases* levantados ou que permitisse enxergar as fronteiras metaforicamente referidas anteriormente.

O objetivo do presente trabalho é somente propor este formalismo, baseado em redes de Petri e seguindo alguns conceitos do Processo Unificado. Será usada a UML representar parte da proposta (os diagramas e especificações de Use Cases) e as redes de Petri para a parte dinâmica que será a verificação dos Requisitos.

São apresentados, na seqüência, os principais conceitos do Processo Unificado, da UML e da rede de Petri.

2.2 Processo Unificado (Unified Process)

Com a crescente demanda para Sistemas de Software cada vez mais complexos, a Engenharia de Software, assim como as demais áreas da Engenharia, precisou definir processos que garantissem a construção de sistemas que:

- Atendam a demanda dos usuários,
- Sejam economicamente viáveis (inclusive sob o ponto de vista do usuário),
- Respeitem o cronograma de entrega do projeto, e
- Possuam um alto índice de qualidade.

Comparando a Engenharia de Software com as demais áreas da Engenharia, verifica-se que esta disciplina é ainda muito jovem. Perante a necessidade de um projeto de construção de uma ponte, verifica-se que os engenheiros da construção civil possuem ampla experiência e, conseqüentemente, processos definidos que podem servir como base para a definição de um novo projeto. Já os engenheiros de software não possuem uma base de conhecimento tão extensa quanto os engenheiros civis e, além disso, de acordo com (Pressman, 1993), "o software não é um produto palpável", assim como uma ponte o é, o que dificulta ainda mais a tarefa de construção de software. Daí a necessidade de um processo para desenvolvimento de sistemas de software que possa tornar essa tarefa mais próxima daquela de se construir uma ponte.

Desde os anos 70 surgiram várias contribuições de autores preocupados com a tarefa de desenvolvimento de Software. Desde a Análise Estrutura (Yourdon, 1989) até as

técnicas baseadas na Orientação a Objetos, pode-se perceber o empenho com que pesquisadores e praticantes de mercado se entregaram à criação de métodos para garantir a qualidade no desenvolvimento de Software (e conseqüentemente de sistemas de natureza semelhantes, como os sistemas automatizados). Na década de 90 surgiu a possibilidade de unificação dessas metodologias, que foi uma proposta (apoiada pelas principais empresas de Software e pelos principais pesquisadores desta área da engenharia) de fusão das várias técnicas existentes até então, para a obtenção de um processo unificado, ou seja, algo que permitisse uma comunicação sem distorções entre diferentes equipes e, conseqüentemente, fosse adotado pela maioria dos profissionais envolvidos na tarefa de desenvolvimento de Software.

Assim surgiu o Processo de Desenvolvimento de Software Unificado, que é um guia que descreve "*Quem* está fazendo *O que*, *Como* e *Quando*" (Jacobson, 1998). Segundo Jacobson este processo:

- Proporciona orientação para ordenar as atividades de uma equipe
- Direciona as tarefas dos desenvolvedores individualmente e da equipe como um todo
- Especifica quais artefatos podem ser desenvolvidos
- Oferece critérios para monitorar e medir os produtos e atividades de um projeto

Descreve-se a seguir as principais características do Processo Unificado, segundo Jacobson:

2.2.1 Baseado em Componentes

O sistema de software construído a partir do Processo Unificado consistirá de componentes de software que podem ser interconectados a partir de interfaces bem definidas.

2.2.2 Dirigido a Use Cases

Use Case pode ser entendido como "o que" o sistema faz para cada usuário (que pode ser pessoas ou máquinas que interagem com o sistema). Baseado no modelo de Use Cases do sistema, os desenvolvedores criam modelos de projeto, implementação e testes que realizam as tarefas dos Use Cases. Por isso diz-se que o Processo Unificado é dirigido a Use Cases.

2.2.3 Centrado na Arquitetura

A arquitetura do software deve ser definida em paralelo com o entendimento do modelo de Use Cases. A medida que os principais Use Cases vão sendo especificados, a arquitetura do software vai se definindo. Para cada Use Case analisado, o mesmo é especificado em detalhes e realizado em termos de sub-sistemas, classes e componentes.

2.2.4 Iterativo e Incremental

Com o objetivo de proporcionar um maior controle das atividades de um projeto, o Processo Unificado propõe que os projetos sejam divididos em "mini projetos", que são as iterações que resultam em um incremento no projeto. As iterações são estrategicamente definidas no planejamento de forma que a necessidade de repetição de uma delas não gere um impacto significativo no prazo ou custo do projeto como um todo.

2.2.5 Ciclo de Vida

O Processo Unificado repete uma série de ciclos durante a vida do sistema. Cada fase concluída corresponde a uma disponibilização do sistema para o cliente. Cada ciclo consiste de quatro fases: Concepção (*Inception*), Elaboração (*Elaboration*), Construção (*Construction*) e Transição (Transition). Cada fase compreende o desenvolvimento de cinco *workflows*: Requisitos (*Requirements*), Análise (*Analysis*),

Projeto (*Design*), Implementação (*Implementation*) e Teste (*Test*). A figura 2-4 ilustra o ciclo de vida do Processo Unificado (RUP, 2001).

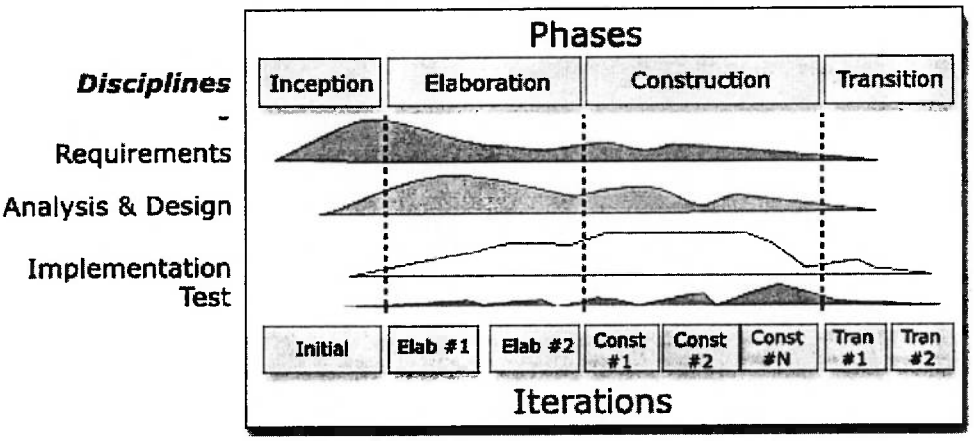


Figura 2-4 Ciclo de Vida do Processo Unificado

2.3 *Unified Modeling Language – UML*

A UML é uma linguagem de modelagem para "especificação, construção e documentação de sistemas de software, bem como para a modelagem de negócios e outros sistemas" (Object Management Group, 2001), e não uma metodologia para desenvolvimento de sistemas.

2.3.1 Histórico da UML

Em Junho de 1996 surgiu a UML 0.9, através de uma iniciativa dos principais pesquisadores sobre metodologias de desenvolvimento de sistemas e da empresas que atuavam na área de desenvolvimento de software. A "UML Consortium", que contava com parcerias importantes tais como HP, Oracle, Microsoft, dentre outras, lançou a UML 1.0 em Janeiro de 1997. Houve então a submissão de alguns trabalhos para padronização à OMG (*Object Management Group*), e em novembro de 1997 a versão 1.1 da UML foi aceita como padrão. A versão atual da UML é a 1.4 e já está em processo de homologação, pela OMG, a versão 2.0 (Object Management Group, 2001).

2.3.2 Objetivos da UML

A UML tem por objetivo padronizar uma notação para especificar, visualizar e documentar o sistema. Trata-se de uma fusão da forma gráfica de representação de modelos dos principais métodos desenvolvidos até o momento, onde se destacam os métodos de Booch (Booch, 1991), OMT (Rumbaugh, 1991) e OOSE (*Object Oriented Software Engineering*) (Jacobson, 1992). De acordo com (Object Management Group, 2001), os objetivos da UML são:

1. Fornecer aos usuários uma linguagem visual expressiva, pronta para o uso no desenvolvimento de modelos de negócios;
2. Fornecer mecanismos de extensibilidade e de especialização para apoiar os conceitos essenciais;
3. Ser independente da linguagem de implementação;
4. Prover uma base formal para entender a linguagem de modelagem;
5. Encorajar o crescimento do número de ferramentas orientadas a objetos;
6. Suportar conceitos de desenvolvimento de níveis mais elevados tais como colaborações, estrutura de trabalho, padrões e componentes;
7. Integrar as melhores práticas de desenvolvimento de software.

2.3.3 Vocabulário

A UML proporciona um vocabulário, que permite a comunicação entre os atores envolvidos no projeto, dividido em três categorias (Jacobson, 1998):

2.3.3.1 Elementos

A primeira categoria possui quatro tipos de elementos, com seus respectivos subtipos:

1. Estrutural (*Structural*)

a. Use case

b. Classe

c. Classe ativa

d. Interface

e. Componente

f. Colaboração

g. Nó

2. Comportamental (*Behavioral*)

a. Interação

b. Máquina de estado

3. Agrupamento (*Grouping*)

a. Pacote

b. Modelo

c. Subsistema

d. Framework

4. Anotações (*Annotational*)

a. Nota

2.3.3.2 Relacionamentos

Existem três tipos de relacionamentos na segunda categoria do vocabulário:

1. Dependência
2. Associação
3. Generalização

2.3.3.3 Diagramas

Na terceira categoria, a UML dispõe de nove tipos de diagramas:

1. Use case
2. Classe
3. Objeto
4. Seqüência
5. Colaboração
6. Statechart
7. Atividade
8. Componente
9. Deployment

2.4 Redes de Petri

O conceito de redes de Petri foi inicialmente proposto por C. A. Petri em 1962, em sua tese de Doutorado, na faculdade de Darmstadt, Alemanha. Petri utilizou-se desse conceito para descrever sistemas de processamento de informações, que se caracterizavam como sendo concorrentes, assíncronos, distribuídos e paralelos. Baseados na formalização matemática de rede de Petri, surgiram métodos de análise da rede, tais como: Árvore de Alcançabilidade, Matriz de Incidência e Equações de Estado, permitindo análises quantitativa e qualitativa do sistema modelado pela rede de Petri (Murata, 1989). Estes métodos não serão detalhados neste trabalho.

2.4.1 Definição

Uma rede de Petri é uma 4-tupla (Murata, 1989), $N = \langle P, T, F, m_0 \rangle$, onde:

- $P = \{p_1, p_2, \ldots, p_n\}$ é um conjunto finito de lugares,

- $T = \{t_1, t_2, \ldots, t_n\}$ é um conjunto finito de transições,
- Satisfazendo $P \cap T = \varnothing$ e $P \cup T \neq \varnothing$
- $F \subseteq (P \times T) \cup (P \times T)$ é um conjunto de arcos orientados (relação de fluxo), e
- m_0: $P \rightarrow \mathbf{IN}$ é a marcação inicial.

Se x e y são elementos genéricos de uma rede de Petri, define-se como pré-conjunto de y ao conjunto dos x tal que o par (x, y) seja uma relação de fluxo da rede. Similarmente, define-se como pós-conjunto de y ao conjunto dos x tal que o par (y, x) esteja incluído nas relações de fluxo

Os símbolos ${}^\bullet p, p^\bullet, {}^\bullet t$, e $t^\bullet$ definem, respectivamente, os pré conjuntos e pós conjuntos de todo lugar p ou transição t.

Uma marcação de uma rede de Petri é a atribuição de um inteiro não negativo a cada lugar. Se o inteiro k é atribuído a um lugar p, diz-se que p está marcado com k marcas. A estrutura de uma rede de Petri define um conjunto de regras de disparo que determinam o comportamento da rede. Uma transição t está habilitada a disparar quando cada $p \in {}^\bullet t$ tiver pelo menos uma marca e se cada lugar $p \in t^\bullet$ tiver lugar para pelo menos uma marca. A rede de Petri caminha de uma marcação para outra pelo disparo das transições habilitadas. Quando uma transição t dispara, uma marca é removida de cada lugar $p \in {}^\bullet t$ e uma marca é adicionada a cada lugar $p \in t^\bullet$. Se m_1 e m_2 são marcações, denota-se por $m_1[t\rangle m_2$ o fato que m_2 é alcançada através de m_1 após a transição t ter sido disparada. Uma marcação m' é dita ser alcançável através de uma marcação m se existir uma seqüência de disparo de transições que transformem m em m'. O conjunto de marcações alcançáveis de m é denotado por $[m\rangle$.

Denota-se por $m(p)$ o número de marcas no lugar p para a marcação m. Também, uma marcação pode ser representada por um vetor de inteiros, $m = (m(p_1), m(p_2), \ldots, m(p_n))$.

Se uma rede de Petri não possuir *self-loop* (conforme figura 2-5), que corresponde ao caminho fechado de um lugar para uma transição e vice-versa, então pode-se utilizar uma matriz de incidência M_i, definida por $M_i = p^{\bullet} - {}^{\bullet}p$, que caracteriza a relação entre os lugares e transições.

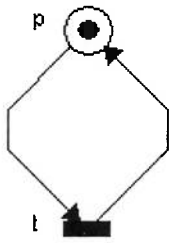


Figura 2-5 Self-loop na rede de Petri

No exemplo ilustrado a seguir, pode-se observar a marcação inicial $m_0 = [1\ 0\ 0\ 0]$, e a respectiva matriz de incidência M_i do sistema, que corresponde a uma matriz n x m, de uma rede de Petri com n transições e m lugares. Cada elemento dessa matriz representa a relação entre lugares e transições. As colunas representam os lugares e as linhas, as transições. O número negativo indica que é um lugar de entrada, o positivo indica que é um lugar de saída e zero indica que não existe relação entre o lugar e a transição correspondente.

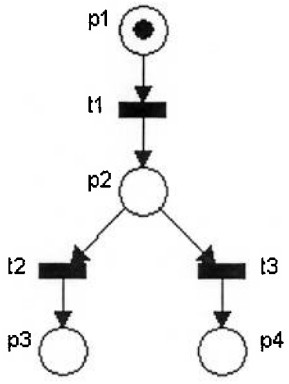


Figura 2-6 Exemplo de rede de Petri

$$A_i = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

Figura 2-7 Matriz de Incidência *p* x *t*

A equação de estado de uma rede de Petri pode ser dada pela seguinte equação:

$$M' = M + A_i\,\sigma$$

onde:

M', é o estado posterior $\rightarrow M' = [0\ 1\ 0\ 0]^T$

M, é o estado atual $\rightarrow M = [1\ 0\ 0\ 0]^T$

A_i, é a Matriz de incidência da rede

σ, é o vetor de disparo $\rightarrow \sigma = [1\ 0\ 0]^T$

2.4.2 Propriedades

As redes de Petri possuem algumas propriedades que são úteis para a análise estrutural e comportamental do sistema em estudo. As principais propriedades são:

- **Consistência**: existe uma marcação inicial e uma seqüência de disparos, de tal forma que após esses disparos, a rede retorna à marcação inicial.
- **Conservabilidade**: para qualquer marcação da rede, a soma das marcas é constante.
- **Limitabilidade**: se o número de marcas em todos os lugares é sempre limitado por algum valor *k*, então a rede é *k*-limitada.
- **Segurança**: uma rede é dita segura quando ela é 1-limitada.
- **Vivacidade**: indica se para uma dada marcação inicial existe, para cada transição, uma seqüência de disparos que leve a uma marcação no qual aquela transição seja habilitada.

2.4.3 Caracterização de Conflito

A situação de conflito é gerada quando um dado lugar p participa do pré-conjunto ou do pós-conjunto de duas ou mais transições diferentes, isto é, dadas as transições habilitadas t_1 e t_2, $p \in {}^\bullet t_1 \cap {}^\bullet t_2$. As situações de conflito representam em geral disputa por recursos ou pela chamada de processos.

Da formalização da rede de Petri, é possível através da estrutura da rede, determinar, para uma dada marcação m_n, quais as transições que estão em conflito. No caso do exemplo ilustrado na figura 2-6, para a marcação m_1 = [0 1 0 0], verifica-se que as transições t_2 e t_3 estão em conflito, pois ambas disputam a retirada da marca do lugar p_2. Formalmente, da matriz de incidência M_i, as linhas que possuírem o mesmo lugar de entrada, ou seja, a coluna que possuir mais de um elemento contendo -1, caracteriza conflito entre as respectivas transições.

3 PROPOSTA DO TRABALHO

Neste capítulo será apresentada a proposta do trabalho que consiste na possibilidade de verificação da consistência lógica do Use Case, simulando o fluxo de eventos da descrição do mesmo. Para tanto, o autor propõe que seja incluída uma atividade de verificação da especificação do Use Case, como sendo uma atividade posterior aquela de "Entender as necessidades do usuário", desenvolvida durante o *workflow* Requisitos (RUP, 2001). Também será considerada a abordagem apresentada no capítulo 2 sobre a validação de requisitos que, de acordo com (Sommerville, 1997), recomenda a visão multidisciplinar do sistema e a tradução dos requisitos em modelos de sistemas. Baseando-se nestas recomendações, será sugerida que a especificação do sistema, ou mesmo a eliciação dos requisitos, seja feita sob, pelo menos, os três pontos de vistas (*viewpoints*) apresentados na figura 2-2 e que os modelos do sistema sejam as respectivas redes de Petri, pois permitem que seja realizada uma verificação da especificação dos requisitos.

No sentido de tornar possível a verificação da consistência do Use Case, será proposta a estruturação da especificação do Use Case na sua descrição do fluxo de eventos. Esta alternativa, além de permitir maior estruturação na representação textual do Use Case, introduz sinais especiais de agrupamento. Estes elementos serão usados mais tarde para balizar a transferência desta informação para redes de Petri (ou poderia ser para um diagrama de Atividades), possibilitando assim a simulação do fluxo dos eventos.

É importante destacar que esta parte do processo é a parte mais sensível da proposta, dado que estará sendo realizada uma transferência semântica de uma representação informal (porém mais disciplinada nesta alternativa) para uma representação formal em redes de Petri, que será utilizada mais tarde para verificação dos requisitos do sistema que está sendo modelado.

A representação em redes de Petri, desta forma, será uma especificação funcional que possibilitará a geração de uma especificação formal sem ambigüidades. Será ilustrada a dualidade entre a especificação da descrição do fluxo de eventos do Use Case e a respectiva rede de Petri.

Para exemplificar, a proposta do trabalho será apresentada através da especificação de uma máquina de auto-atendimento bancário, mais conhecida pela sigla em Inglês ATM (Automated Teller Machine).

3.1 Verificação da Especificação de Requisitos

O autor propõe que seja incluída, explicitamente, uma atividade que possibilite a "Verificar a consistência dos Requisitos do Usuário", que é a garantia de que eliciação das necessidades dos atores envolvidos no sistema estará de acordo com o que o ator acredita ter especificado. Esta atividade seria incluída no *workflow* de Requisitos do Processo Unificado visto no capítulo 2. A figura 3-1 ilustra a localização desta atividade no novo diagrama de Atividades do *workflow* de Requisitos do Processo Unificado, que já incluiria a atividade proposta.

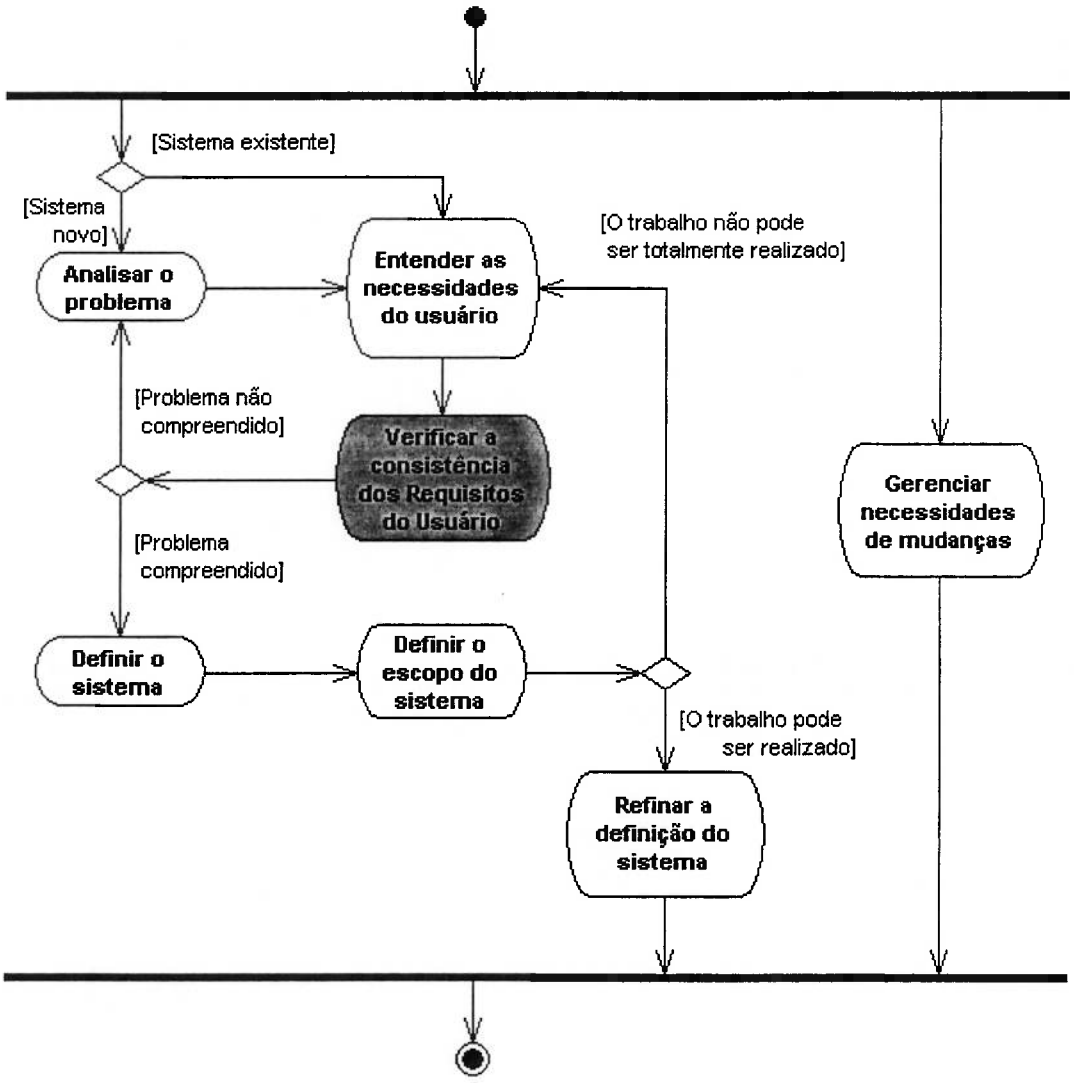


Figura 3-1 Novo diagrama de atividades para o *workflow* de Requisitos

Desta forma, após a atividade de "Entender as necessidades do usuário", o analista realiza a atividade de "Verificar a consistência dos Requisitos do Usuário". O autor acredita que esta verificação é importante porque permite verificar se o entendimento da necessidade do usuário condiz com o que o usuário (e/ou outros atores importantes na definição do sistema) acredita que o sistema permitirá realizar.

A seguir o autor justifica, com base no levantamento efetuado e na sua pequena experiência no desenvolvimento de software, o principal objetivo desta proposta que é permitir a redução de erros nas fases iniciais do desenvolvimento de software.

Uma das contribuições do método OOSE (*Object Oriented Software Engineering*) (Jacobson, 1992) foi favorecer a fase inicial de desenvolvimento, isto é, a análise de requisitos. Esta contribuição foi fundamental para minimizar as incompreensões entre analistas e usuários, através da descrição dos Use Cases, tornando o levantamento de necessidades do sistema um exercício de descrição funcional.

O modelo de Use Cases torna objetiva a apresentação do entendimento da análise de requisitos, uma vez que os símbolos utilizados representam univocamente as partes do sistema, ou seja, as atividades que serão desempenhadas pelos atores do sistema (aqui entendidos como pessoa, máquina ou outro sistema). Essa aproximação, também conhecida como "*Use Case Driven*" (Jacobson, 1998), por um lado reduziu a distância entre a Análise de Negócios (do contexto em que o sistema de software seria utilizado) e a Análise de Requisitos, porém gerou o aumento da "distância" entre a Análise de Requisitos e o Design de Sistemas, mesmo para artefatos simples.

No caso dos sistemas mecatrônicos, também se percebe o problema semelhante ao do desenvolvimento de software em geral. Assim sendo, os sistemas mecatrônicos também perceberam um aumento na demanda para a integração entre controle, gestão e apoio a decisão, combinando a integração horizontal dos processos de chão de fábrica (através de sistemas de supervisão e de controle discreto) com a integração vertical dos processos de gestão, controle de estoque e decisão gerencial. Pode-se chamar a estes sistemas (aplicáveis não somente à manufatura) de sistemas de informação.

Por outro lado, o aumento da interatividade e confiança neste novo sistema de informação demanda um desenvolvimento muito mais acurado, onde o papel de todos os atores (humanos, máquinas, computadores, etc.) tem que ser previsto desde o início e deve ser respeitado durante todo o processo, que vai desde a especificação até a implementação.

Neste trabalho dedicou-se especial atenção às fases iniciais do desenvolvimento, já que acumulam a maior taxa de erros (ou os erros de maior impacto). É também nesta fase onde a correção dos erros tem custo menor. A figura 3-2 (Sanders, 2001) mostra a evolução do custo de correção dos erros de projeto com o avanço do ciclo de vida.

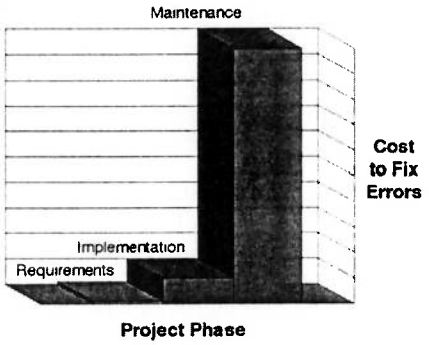


Figura 3-2 Custo para corrigir erros X Fase do Projeto

Notadamente, a maior parte das ferramentas computacionais desenvolvidas (Ghezzi, 1991) (Queens's University, 2002), seja pelos pesquisadores da área de Modelagem e Design, seja pelos pesquisadores da área de Engenharia de Software, é dedicada aos níveis mais avançados do ciclo de vida, isto é, projeto e implementação. Uma razão para isto, provavelmente se deve ao fato de que estas fases englobam representações formais que podem ser comparadas, verificadas, analisadas (analítica ou computacionalmente), simuladas e testadas funcionalmente. As fases iniciais do processo, ao contrário, envolvem uma passagem de uma fase informal de levantamento de requisitos (e que precisa ser informal nesta fase) para uma fase mais disciplinada e formal de especificação.

A proposta apresentada aqui é justamente para disciplinar a fase de levantamento de requisitos (sem contudo tentar formalizar ou tornar rígida a descrição funcional do novo artefato) e através de um processo menos informal, passar de uma representação livre para uma representação formal em redes de Petri, que através de um processo evolutivo se transformará em uma especificação funcional executável, isto é, passível de verificação computacional.

Para tanto, recorre-se à Engenharia de Software, particularmente ao Processo Unificado, não apenas por este ser a tendência atual na área de software, mas pelas seguintes razões:

- trata-se de um processo onde os requisitos são levantados por funcionalidade e pelo comportamento do novo artefato

- é um processo adaptado para o desenvolvimento de sistemas de grande porte, com uma multiplicidade de agentes, genericamente chamados de "atores"
- pressupõe-se tratar de sistemas que têm dinâmica própria e/ou em que a melhor descrição do sistema em questão está na associação entre sua dinâmica e a descrição individual e paramétrica dos seus atores

O modelo de representação da descrição de Use Cases do Processo Unificado prevê tais aspectos, porém existe ainda uma distância entre a forma de representação sugerida por este e uma que possa ser tratada computacionalmente de modo a gerar metáforas cada vez mais formais do mesmo comportamento.

Na tentativa de diminuir essa distância, foi proposta uma alteração na forma de descrever o Fluxo de Eventos dos Use Cases, o que pode aumentar o grau de entendimento da seqüência das atividades desempenhadas pelo Use Case, tanto do fluxo principal como do fluxo alternativo dos eventos. Outro benefício, talvez o principal deles, é que a introdução de símbolos (poderia ser uma outra forma de identificação) na descrição dos eventos pode, por exemplo, permitir que redes de Petri possam ser geradas automaticamente a partir dessa descrição do Fluxo de Eventos do Use Case.

O próximo passo seria validar a modelagem ainda na fase de análise de requisitos. Isto significa traduzir este modelo inicial em uma representação mais formal, por exemplo, uma metáfora deste em redes de Petri. Por metáforas (Silva, 1998), entenda-se uma nova representação com o mesmo conteúdo semântico mas em outro "*framework*".

Para que a atividade proposta pelo autor, "Verificar a consistência dos Requisitos do Usuário", seja possível, primeiramente se faz necessária a apresentação da estruturação da descrição do fluxo de eventos do Use Case que estará sendo especificado.

Para uma apresentação mais clara da proposta, será utilizado um exemplo clássico de Use Case, e que também atende às características do tipo de sistema que é descrito

neste capítulo: o Saque de Dinheiro (RUP, 2001), isto é, um exemplo de máquina ATM para serviço bancário personalizado.

3.2 Estruturação da Representação do Fluxo de Eventos do Use Case

Genericamente, os requisitos de um sistema compõem "o que o sistema faz", isto é, uma descrição funcional da atividade ou comportamento principal do sistema. Por analogia com a nomenclatura do CIMOSA (Vernadat, 1996), esta funcionalidade será chamada de *business process*.

Portanto, *business process* é uma seqüência de eventos cujas propriedades desejáveis são:

- que seja um conjunto próprio de eventos, isto é, com uma única entrada e uma única saída, e que todos os elementos pertençam a um dos caminhos que levam da entrada à saída;
- que este processo principal (*business process*) seja único;

O *business process* é composto de eventos atômicos chamados (ainda utilizando a nomenclatura do CIMOSA) *enterprise activities*. Uma *enterprise activity* pode ser entendida como uma operação de chão de fábrica, seja de armazenamento, transporte, transformação, ou ainda, de projeto do produto.

No fluxo principal de atividade podem ocorrer, condicionalmente, operações alternativas (*enterprise activities*) visando a correção do processo. Por exemplo, uma dada peça que passa por sucessivos processos de transformação pode ser rejeitada em um ponto de teste e retornar para re-manufatura ou ser simplesmente descartada como refugo. É possível encarar estes pontos de teste e desvio condicional como operações alternativas.

Estes processos estão plenamente de acordo com o Use Case onde os elementos principais fazem parte do processo principal e as operações alternativas constituem o fluxo alternativo, como é mostrado no exemplo a seguir. No caso de sistemas

automatizados esta dualidade é ainda mais importante, dado que neste caso se supõe uma certa autonomia de funcionamento.

Para o exemplo que será descrito no item 3.2.1, será considerado o modelo de Use Cases da figura 3-3, sendo detalhado o Use Case "Saque de Dinheiro" (RUP, 2001).

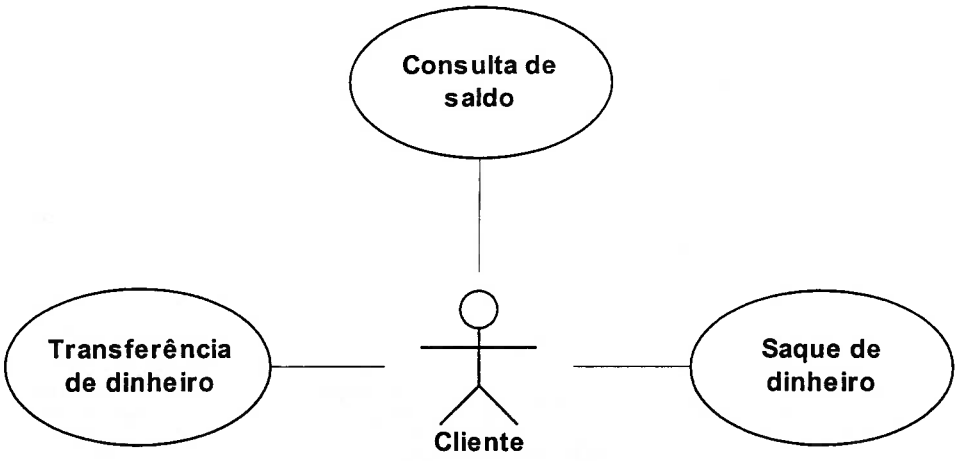


Figura 3-3 Modelo de Use Cases da ATM

O autor considera que a especificação do sistema tende a possuir menos erros se forem observados, pelo menos, os três pontos de vista da figura 2-2. Portanto, a especificação do use case Saque de Dinheiro será dividida, inicialmente de acordo o modelo de descrição de fluxo de evento observado em (RUP, 2001), em três pontos de vista: do Usuário final, do Patrocinador do sistema e do Analista. Finalmente, será apresentada a proposta de estruturação da descrição do fluxo de eventos do Use Case como sendo uma fusão dos diferentes pontos de vista do use case Saque de Dinheiro da ATM.

3.2.1 Ponto de vista do Patrocinador do Sistema

Segundo o ponto de vista do patrocinador do sistema, o use case Saque de Dinheiro deve possuir os seguintes fluxos de eventos:

3.2.1.1 Fluxo Principal de Eventos

1. Inicia Saque – O Cliente insere o cartão do banco na leitora de cartões da máquina ATM.
2. Solicita Senha – A ATM solicita a senha do cliente (4 dígitos)
3. Verifica código da conta e senha – O código da conta e senha são verificados para determinar se a conta é válida e se a senha informada é a senha correta para a conta. Nesse ponto, a conta é uma conta válida e a senha é a senha correta associada à conta.
4. Opções da ATM – A ATM mostra as diferentes opções disponíveis nessa ATM. Nesse ponto, o cliente do banco sempre seleciona "Saque de Dinheiro".
5. Solicita Valor – A ATM solicita o valor do saque. Nesse ponto, o cliente seleciona os valores pré estabelecidos ($10, $20, $50, ou $100).
6. Autorização – A ATM inicia o processo de verificação com o Sistema do Banco enviando o ID do cartão, a senha, valor e informações da conta como uma transação. Nesse ponto, o Sistema do Banco está on-line e retorna com a autorização para completar o saque com sucesso, atualizando a conta de acordo o saque.
7. Dispensa dinheiro – O Dinheiro é dispensado.
8. Emite Recibo – O recibo é impresso e dispensado quando houve saque.

3.2.1.2 Fluxo Alternativo de Eventos

1. ATM sem Dinheiro – No evento 4 do Fluxo principal – Opções da ATM, se a ATM estiver ser dinheiro, a opção "Saque de Dinheiro" não estará habilitada.
2. Dinheiro insuficiente na ATM – No evento 5 do Fluxo principal – Solicita Valor, se a ATM não contém fundos suficiente para atender o valor solicitado, uma mensagem será mostrada e ocorre o retorno para o evento 5 do Fluxo principal – Solicita Valor.

3. Senha Incorreta – No evento 3 do Fluxo principal – Verifica o código da conta e a senha, o cliente tem três tentativas para informar a senha correta. Se a senha informada for incorreta, a ATM exibirá uma mensagem apropriada e se restarem mais tentativas o fluxo volta para o evento 2 do Fluxo principal – Solicita senha. Se, ao final das tentativas a senha informada continuar incorreta, o cartão será retido.

4. Saldo insuficiente na conta – No evento 6 do Fluxo principal – Autorização, o sistema do banco retorna um código informando que o saldo da conta é menor que o valor informado no evento 5 – Solicita Valor, a ATM exibirá uma mensagem apropriada e o fluxo voltará no evento 5 do Fluxo principal – Solicita Valor.

5. Valor excede limite de saque diário – No evento 5 do Fluxo principal – Solicita Valor, o sistema do banco retorna um código indicando que, incluindo essa solicitação de saque, o cliente excedeu ou excederá o limite permitido no período de 24 horas, a ATM exibirá uma mensagem apropriada e o fluxo voltará no evento 5 do Fluxo principal – Solicita Valor.

6. Sair – O cliente pode, a qualquer momento, decidir cancelar a transação (sair).

7. "*Tilt*" – A ATM possui diversos sensores que monitoram diferentes funções, tais como energia elétrica, pressão exercida nos vários compartimentos que possam ser abertos e sensores de movimento. Se, a qualquer momento, um sensor for ativado, um sinal de alarme é enviado para a Polícia e a ATM entrará em "Modo Seguro" onde todas as funções estarão suspensas até que seja executada uma ação de reinicialização.

3.2.2 Ponto de vista do Usuário final do Sistema

Segundo o ponto de vista do usuário final do sistema, o use case Saque de Dinheiro deve possuir os seguintes fluxos de eventos:

3.2.2.1 Fluxo Principal de Eventos

1. Inicia Saque – O Cliente insere o cartão do banco na leitora de cartões da máquina ATM.
2. Solicita Senha – A ATM solicita a senha do cliente (4 dígitos)
3. Solicita Valor – A ATM solicita o valor do saque. Nesse ponto, o cliente seleciona os valores pré estabelecidos ($10, $20, $50, ou $100).
4. Dispensa dinheiro – O Dinheiro é dispensado.
5. Emite Recibo – O recibo é impresso e dispensado quando houve saque.

3.2.2.2 Fluxo Alternativo de Eventos

1. Senha Incorreta – No evento 2 do Fluxo principal – V Solicita senha, o cliente tem três tentativas para informar a senha correta. Se a senha informada for incorreta, a ATM exibirá uma mensagem apropriada e se restarem mais tentativas o fluxo volta para o evento 3 do Fluxo principal – Solicita senha. Se, ao final das tentativas a senha informada continuar incorreta, o cartão será retido.
2. Valor excede limite de saque diário – No evento 3 do Fluxo principal – Solicita Valor, o sistema do banco retorna um código indicando que, incluindo essa solicitação de saque, o cliente excedeu ou excederá o limite permitido no período de 24 horas, a ATM exibirá uma mensagem apropriada e o fluxo voltará no evento 3 do Fluxo principal – Solicita Valor.

3.2.3 Ponto de vista do Analista do Sistema

Finalmente, segundo o ponto de vista do analista do sistema, o use case Saque de Dinheiro deve possuir os seguintes fluxos de eventos:

No início deste Use Case a ATM está no estado Pronto.

1. Inicia Saque – O Cliente insere o cartão do banco na leitora de cartões da máquina ATM.

2. Verifica cartão do banco – A ATM lê o código da conta da tarjeta magnética do cartão do banco e checa se o cartão do banco é válido.
3. Solicita Senha – A ATM solicita a senha do cliente (4 dígitos)
4. Verifica código da conta e senha – O código da conta e senha são verificados para determinar se a conta é válida e se a senha informada é a senha correta para a conta. Nesse ponto, a conta é uma conta válida e a senha é a senha correta associada à conta.
5. Solicita Valor – A ATM solicita o valor do saque. Nesse ponto, o cliente seleciona os valores pré estabelecidos ($10, $20, $50, ou $100).
6. Autorização – A ATM inicia o processo de verificação com o Sistema do Banco enviando o ID do cartão, a senha, valor e informações da conta como uma transação. Nesse ponto, o Sistema do Banco está on-line e retorna com a autorização para completar o saque com sucesso, atualizando a conta de acordo o saque.
7. Dispensa dinheiro – O Dinheiro é dispensado.
8. Emite Recibo – O recibo é impresso e dispensado quando houve saque.
9. Devolve Cartão – O cartão do banco é devolvido. A ATM também atualiza o *log* interno de acordo a operação.

   O Use Case termina e a ATM está no estado Pronto.

3.2.3.1 Fluxo Alternativo de Eventos

1. Cartão inválido – No evento 2 do Fluxo principal – Verifica cartão do banco, se o cartão for inválido, o mesmo é devolvido e a máquina emite uma mensagem apropriada.
2. Dinheiro insuficiente na ATM – No evento 5 do Fluxo principal – Solicita Valor, se a ATM não contém fundos suficiente para atender o valor solicitado, uma

mensagem será mostrada e ocorre o retorno para o evento 5 do Fluxo principal – Solicita Valor.

3. Senha Incorreta – No evento 4 do Fluxo principal – Verifica o código da conta e a senha, o cliente tem três tentativas para informar a senha correta. Se a senha informada for incorreta, a ATM exibirá uma mensagem apropriada e se restarem mais tentativas o fluxo volta para o evento 3 do Fluxo principal – Solicita senha. Se, ao final das tentativas a senha informada continuar incorreta, o cartão será retido, a ATM retorna para o estado Pronto e o use case termina.

4. Conta inválida – No evento 4 do Fluxo principal – Verifica conta e senha, se o sistema do Banco retorna um código indicando que a conta não pode ser encontrada ou não é um conta que permite saques, a ATM exibirá uma mensagem apropriada e o fluxo continuará no evento 8 do Fluxo principal – Devolve Cartão.

5. Saldo insuficiente na conta – No evento 6 do Fluxo principal – Autorização, o sistema do banco retorna um código informando que o saldo da conta é menor que o valor informado no evento 5 – Solicita Valor, a ATM exibirá uma mensagem apropriada e o fluxo voltará no evento 5 do Fluxo principal – Solicita Valor.

6. Valor excede limite de saque diário – No evento 5 do Fluxo principal – Solicita Valor, o sistema do banco retorna um código indicando que, incluindo essa solicitação de saque, o cliente excedeu ou excederá o limite permitido no período de 24 horas, a ATM exibirá uma mensagem apropriada e o fluxo voltará no evento 5 do Fluxo principal – Solicita Valor.

7. Erro de *Log* – Se no evento 9 do Fluxo principal – Recibo, o *log* não pode ser atualizado, a ATM entrará em "Modo Seguro" onde todas as funções estarão suspensas. Um alarme apropriado é enviado para o sistema do Banco para indicar que a operação da ATM foi suspensa.

8. Sair – O cliente pode, a qualquer momento, decidir cancelar a transação (sair). A transação será suspense e o cartão devolvido.

9. "*Tilt*" – A ATM possui diversos sensores que monitoram diferentes funções, tais como energia elétrica, pressão exercida nos vários compartimentos que possam ser abertos e sensores de movimento. Se, a qualquer momento, um sensor for ativado, um sinal de alarme é enviado para a Polícia e a ATM entrará em "Modo Seguro" onde todas as funções estarão suspensas até que seja executada uma ação de reinicialização.

Como se pode ver do exemplo acima, o modelo (que é o convencional) propõe uma especificação de processos linear, em camadas, onde primeiramente se representa o primeiro nível de processamento e as opções são as canonicamente esperadas, sem procedimentos de exceção. Depois representa-se o segundo nível, com as exceções ao primeiro e assim por diante.

Para sistemas grandes que demandem processos longos (embora estes não sejam recomendáveis na Engenharia de Software), a separação entre a causa do procedimento de exceção e seu corpo pode trazer problemas, além de perder a noção de estruturação (o conceito de bloco funcional). Deve-se ressaltar que a ocorrência de processos longos na Engenharia está longe de ser um problema de estilo do projetista, principalmente para os sistemas de informação modernos.

Portanto, faz-se necessário uma alternativa à simplicidade da descrição do fluxo de eventos do Use Case apresentado.

A seguir é apresentada a introdução de marcações de bloco funcional e um direcionamento mais claro do fluxo dos eventos para a descrição dos Use Cases.

3.2.4 Símbolos acrescentados à descrição do Fluxo de Eventos

Com o objetivo de permitir que a partir da descrição do fluxo de eventos do use case seja possível verificar a consistência lógica das atividades desempenhadas pelo use case, foi necessária a introdução dos símbolos reservados a seguir:

Símbolo	Descrição

●	Início de um processo (Use Case)
◉	Fim de um processo (Use Case)
→	Início de um evento do fluxo principal do processo
↳	Início de um evento do fluxo alternativo ao fluxo principal do processo
◇	Início de um evento condicional
~n	Desvio da iteração corrente para a iteração n
||	Seqüência de eventos concorrentes

Tabela 3-1 Símbolos adicionados na descrição do Use Case

Desta forma, é exemplificado a seguir o impacto desta disciplina (que em nada impõe restrições à liberdade dos Use Cases) no mesmo exemplo para o use case de Saque de Dinheiro em máquina ATM, porém como sendo um ponto de vista consolidado do sistema, mesclando os pontos de vista ilustrados anteriormente: o ponto de vista do Usuário final, do Patrocinador do sistema e do Analista.

3.2.4.1 Fluxo Principal de Eventos com Fluxos Alternativos

1 ● Inicia Saque – O Cliente insere o cartão do banco na leitora de cartões da máquina ATM; (

1.1 ◇ Cartão é Válido? – Verifica Cartão do Banco – A ATM lê o código da conta da tarjeta magnética do cartão do banco e checa se o mesmo é válido;

1.1.1 ↳ Emite mensagem Cartão Inválido – Se o cartão for inválido, o mesmo é devolvido e a máquina emite uma mensagem apropriada. ~1.9

1.2 → Solicita Senha – A ATM solicita a senha do cliente (4 dígitos);

1.3 ◇ Senha Correta e Conta válida? – Verifica Código da Conta e Senha – O código da conta e senha são verificados para determinar se a conta é válida e se a senha informada é a senha correta para a conta;

1.3.1 ◇ Conta Válida? – Verifica Código da Conta – O código da conta é verificado;

1.3.1.1 ↳ Emite mensagem Conta Inválida – O sistema do Banco retorna um código indicando que a conta não pode ser encontrada ou que não é uma conta que permite saques. ~1.9

1.3.2 ◇ Número de tentativas >= 3? – Verifica número de tentativas – O cliente tem três tentativas para informar a senha correta;

1.3.2.1 ↳ Emite mensagem Senha Incorreta – A ATM exibirá uma mensagem. ~1.2

1.3.3 ↳ Emite mensagem Excedeu número de tentativas – Ao final das tentativas o cartão será retido e a ATM retorna para o estado pronto. ~2

1.4 ◇ ATM possui Dinheiro? – Seleciona Saque – A ATM mostra as diferentes opções disponíveis. Nesse caso o cliente do banco sempre seleciona "Saque de Dinheiro";

1.4.1 ↳ Emite mensagem ATM sem Dinheiro – Se a ATM estiver ser dinheiro, a opção "Saque" não estará habilitada. ~1.4

1.5 ◇ Valor Suficiente e não Excede Limite? – Solicita Valor – A ATM solicita o valor do saque. Nesse caso o cliente seleciona os valores pré estabelecidos ($10, $20, $50, ou $100);

1.5.1 ◇ Valor suficiente? – Verifica valor suficiente – Verifica se existe dinheiro suficiente;

1.5.1.1 ↳ Emite mensagem Valor Insuficiente na ATM – A ATM não possui dinheiro suficiente para atender ao valor. ~1.5

1.5.2 ↳ Emite mensagem Valor Excede Limite de Saque Diário – O sistema do banco retorna um código indicando que, incluindo essa solicitação de saque, o cliente excedeu ou terá excedido o limite permitido no período de 24 horas. ~1.5

1.6 ◇ Saque Autorizado? – Autoriza saque – A ATM inicia o processo de verificação com o Sistema do Banco enviando o ID do cartão, a senha, valor e informações da conta como uma transação. Nesse ponto, o Sistema do Banco está on-line e retorna com a autorização para completar o saque com sucesso, atualizando a conta de acordo o saque;

1.6.1 ↳ Emite mensagem Saldo Insuficiente na Conta – O sistema do banco retorna um código informando que o saldo da conta é insuficiente. ~1.5

1.7 → Dispensa dinheiro – O Dinheiro é dispensado;

1.8 → Emite Recibo – O recibo é impresso e dispensado quando houve saque;

1.9 → Devolve Cartão – O cartão do banco é devolvido;

1.10 ◇ Operação Concluída? – Atualiza *log* – A ATM também atualiza o *log* interno de acordo a operação;

1.10.1 ↳ Suspende Operação – O *log* não pode ser atualizado, a ATM entrará em "Modo de Segurança". Um alarme apropriado é enviado para o sistema do Banco indicando que a operação da ATM foi suspensa. ~2

) 2 ◉ Finaliza Operação – O Use Case termina com a ATM no estado Pronto;

3.2.4.2 Fluxo Alternativo de Eventos

> 3 ↳ Cancela Operação – O cliente pode, a qualquer momento, decidir cancelar a transação (sair). A transação será suspensa e o cartão devolvido. ⇝1.9
>
> 4 ↳ Ativa sensor de "*Tilt*" – A ATM possui diversos sensores que monitoram diferentes funções, tais como energia elétrica, pressão exercida nos vários compartimentos que possam ser abertos e sensores de movimento. Se, a qualquer momento, um sensor for ativado, um sinal de alarme é enviado para a Polícia e a ATM entrará em "Modo Seguro" onde todas as funções estarão suspensas até que seja executada uma ação de reinicialização. ⇝2

Até visualmente as identações mostram os blocos alternativos, onde eventualmente o controle do processo é redirecionado para outra iteração posterior ou anterior ao ponto corrente. O fluxo alternativo tem outra conceituação e é redirecionado aos eventos incondicionais ocorridos em situações especiais. Neste caso, se representa estes fluxos como outros processos que são iniciados, eliminando os anteriores.

Portanto, na representação proposta há uma diferenciação clara entre uma iteração alternativa (com possibilidade de retomar o controle do processo no mesmo ponto ou redirecionando o fluxo para outro ponto) e um novo processo que elimina o processo corrente, pressupondo um sistema pré-emptivo.

Na seqüência será apresentada a estruturação da representação do fluxo de eventos de um Use Case de forma a permitir que um interpretador simples possa gerar uma rede de Petri equivalente ao fluxo de eventos.

3.3 Conversão da representação do Use Case em rede de Petri

A introdução de símbolos especiais na representação informal do Use Case a torna mais disciplinada além de permitir realizar uma transferência semântica para uma representação formal em redes de Petri.

Baseado na notação BNF (*Backus Naur Form*) (Naur, 1960), inicialmente é definida a estrutura semântica para os fluxos de eventos da descrição do Use Case. Posteriormente será tratada a interpretação dessa estrutura para se gerar a rede de Petri equivalente.

3.3.1 Definição da Estrutura da Descrição do Use Case

A necessidade de se gerar uma especificação formal a partir da descrição do Use Case torna imprescindível a definição de uma estrutura semântica para se descrever Use Cases.

Portanto, seja FP o conjunto de eventos do fluxo principal da descrição do Use Case e FA o conjunto de eventos do fluxo alternativo da descrição do Use Case. Dessa forma a estrutura da descrição do Use Case, segundo a notação BNF, pode ser definida como segue.

```
<FP>          ::= <inicial> { <evento> } <final> { <FA> }

<inicial>     ::= <label> "●" <título> "-" <descrição> ";" "("

<label>       ::= <número> { "." <número> }

<número>      ::= <número_sequencial>

<título>      ::= <string>

<descrição>   ::= <string>

<string>      ::= <qualquer_caracter> { <qualquer_caracter> }

<evento>      ::= [ <eventoP> | <eventoA> ]

<eventoP>     ::= <labelP> [ ( "→" <título> "-" <descrição> ";" ) |

                             ( "◇" <condição> "-" <título> "-" <descrição>
                             ";" <eventoA> ) ] { <eventoP> } |

                  "||" "(" <eventoP> { <eventoP> } ")" "(" <eventoP> {
                  <eventoP> } ")" "||" { <eventoP> }
```

```
<labelP>      ::= <label> "." <número>

<condição>    ::= <string>

<eventoA>     ::= <labelA> [ ( "↳" <título> "-" <descrição> <desvio> ) |

                             ( "◇" <condição> "-" <título> "-" <descrição>
                             ";" <eventoA> ) ] { <eventoA> }

<labelA>      ::= <labelP> "." <número>

<desvio>      ::= "~" [ <label> | <labelP> ]

<final>       ::= ")" <label> "⊙" <título> "-" <descrição> ";"

<FA>          ::= <eventoAI>

<eventoAI>    ::= <label> "↳" <título> "-" <descrição> <desvio> { <eventoAI>
                  }
```

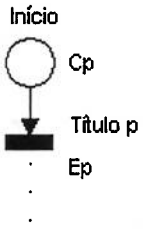
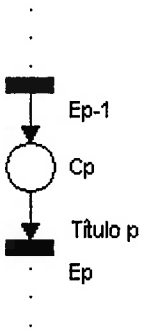
onde:

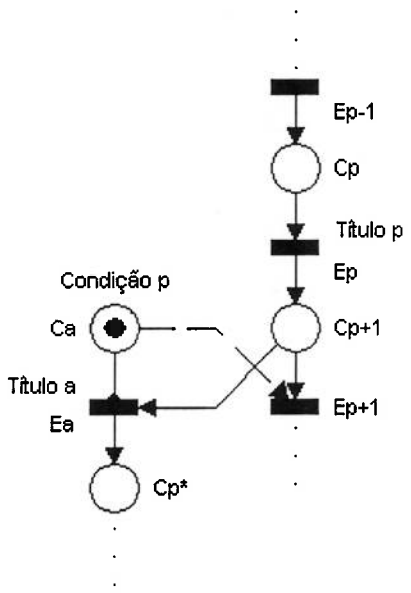
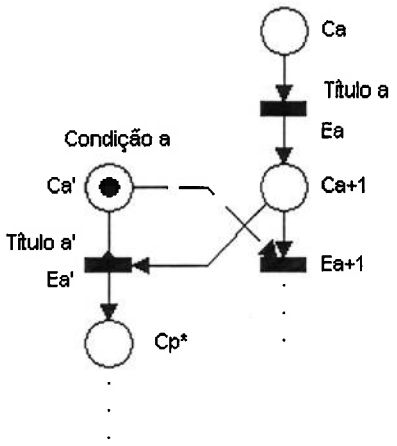
- eventoP → Evento do fluxo principal
- labelP → Identificador do evento do fluxo principal
- eventoA → Evento alternativo do fluxo principal
- labelA → Identificador do evento alternativo do fluxo principal
- eventoAI → Evento incondicional do fluxo alternativo
- labelAI → Identificador do evento incondicional do fluxo alternativo

A definição acima permite gerar metáforas da descrição proposta do Use Case, que ainda podem ser entendidas como uma representação de requisitos, mas ao mesmo tempo com uma estrutura que pode ser facilmente "entendida" por um interpretador que poderia operar a transformação desta para uma rede de Petri elementar. Para tanto é preciso formalizar este processo de transformação da nova representação de Use Case para uma nova metáfora em Redes de Petri.

3.3.2 Segmento da rede de Petri equivalente à descrição do Use Case

Da definição anterior (representação da descrição do Use Case na notação BNF), pode-se introduzir a especificação de um interpretador que geraria a metáfora da descrição do Use Case em redes de Petri. Para garantir a transferência semântica apropriada entre as representações, foi realizado uma mapeamento sintático entre o Use Case e trechos de rede. Este mapeamento garante, por construção, a metáfora agora expressa na linguagem de redes de Petri.

Evento	Notação BNF	Segmento da rede de Petri equivalente
Inicial	`P ● Título - Descrição; (`	
Principal normal	`P → Título - Descrição;`	

Principal condicional e alternativo normal[1]	P◇Condição - Título - Descrição; a ↳ Título - Descrição ~P*	
Alternativos condicional e normal	a◇Condição - Título - Descrição; a' ↳ Título - Descrição ~P*	

[1] A condição Ca aqui representa uma condição externa, onde o arco de Ca para o evento Ea é um arco inibidor, enquanto que o arco de Ca para Ep+1 é um arco habilitador.

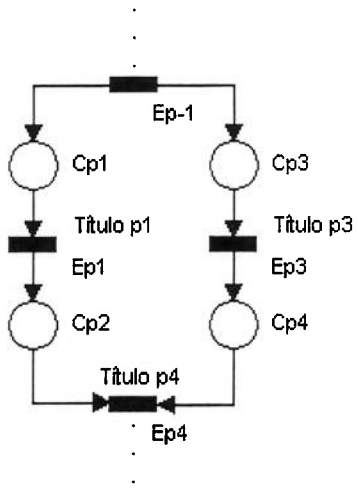
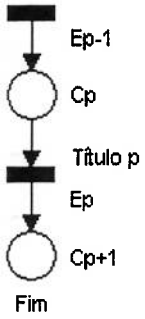
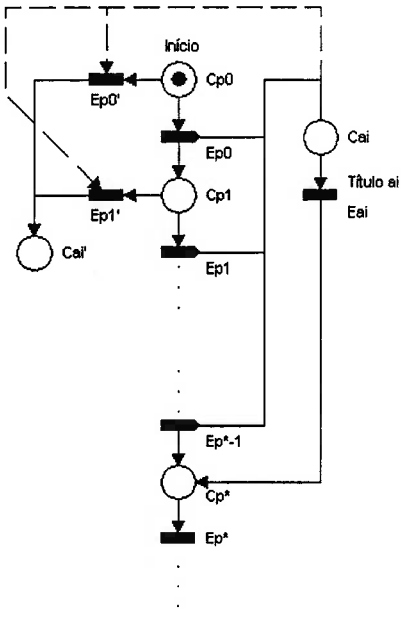
Principal concorrente	|| (p_1 → Título - Descrição; p_2 → Título - Descrição;) (p_3 → Título - Descrição; p_4 → Título - Descrição;) ||	
Final	) p ◉ Título - Descrição;	
Alternativo incondicional	ai ↳ Título - Descrição ~p*	

Tabela 3-2 Rede de Petri equivalente à descrição dos Eventos

O esquema acima tem como principal objetivo permitir que uma ferramenta de validação de requisitos seja gerada, de modo a permitir:

1. que os requisitos sejam visualizados e verificados em um ambiente de simulação
2. a detecção e eliminação de ambigüidades nos requisitos do sistema, ainda na fase de análise de requisitos, minimizando assim custos e trabalhos desnecessários
3. que o entendimento das regras de negócios (*business process*) sejam rapidamente compreendidos pelos analistas de negócio, uma vez que os modelos a serem verificados são gerados automaticamente a partir da descrição em linguagem natural, ou até mesmo a partir de diagrama de Atividades
4. a diminuição do número das iterações no processo de desenvolvimento do sistema em questão pelo simples fato de que a validação dos requisitos do sistema já permite perceber detalhes no modelo de negócios que, segundo o processo sugerido pelo Processo Unificado, na maioria dos casos só seriam percebidos em iterações futuras
5. que outros modelos UML possam ser gerados automaticamente a partir da descrição do Use Case, como por exemplo os modelos dinâmicos: Diagrama de Atividades e Diagrama de Estados

A partir da especificação preliminar é possível passar à uma versão mais formal através dos conhecidos métodos de modelagem e análise de sistemas discretos em redes de Petri. A vantagem, neste caso, é poder tratar sempre com o comportamento do sistema e suas propriedades.

3.3.3 Rede de Petri equivalente à descrição proposta do Use Case

Para ilustrar a aplicação da proposta de estruturação na descrição de Use Cases, foi feita a modelagem em redes de Petri da descrição do Use Case do item 3.2.4.

Elaborou-se, a partir da definição do item 3.3.2, a rede de Petri equivalente à descrição do Use Case do item 3.1.2. Para tanto, foi utilizado o HPSim (Anschuetz, 2001), ferramenta para edição e simulação de redes de Petri.

O modelo, depois de transformado e refinado, pode ser visualizado na figura 3-4.

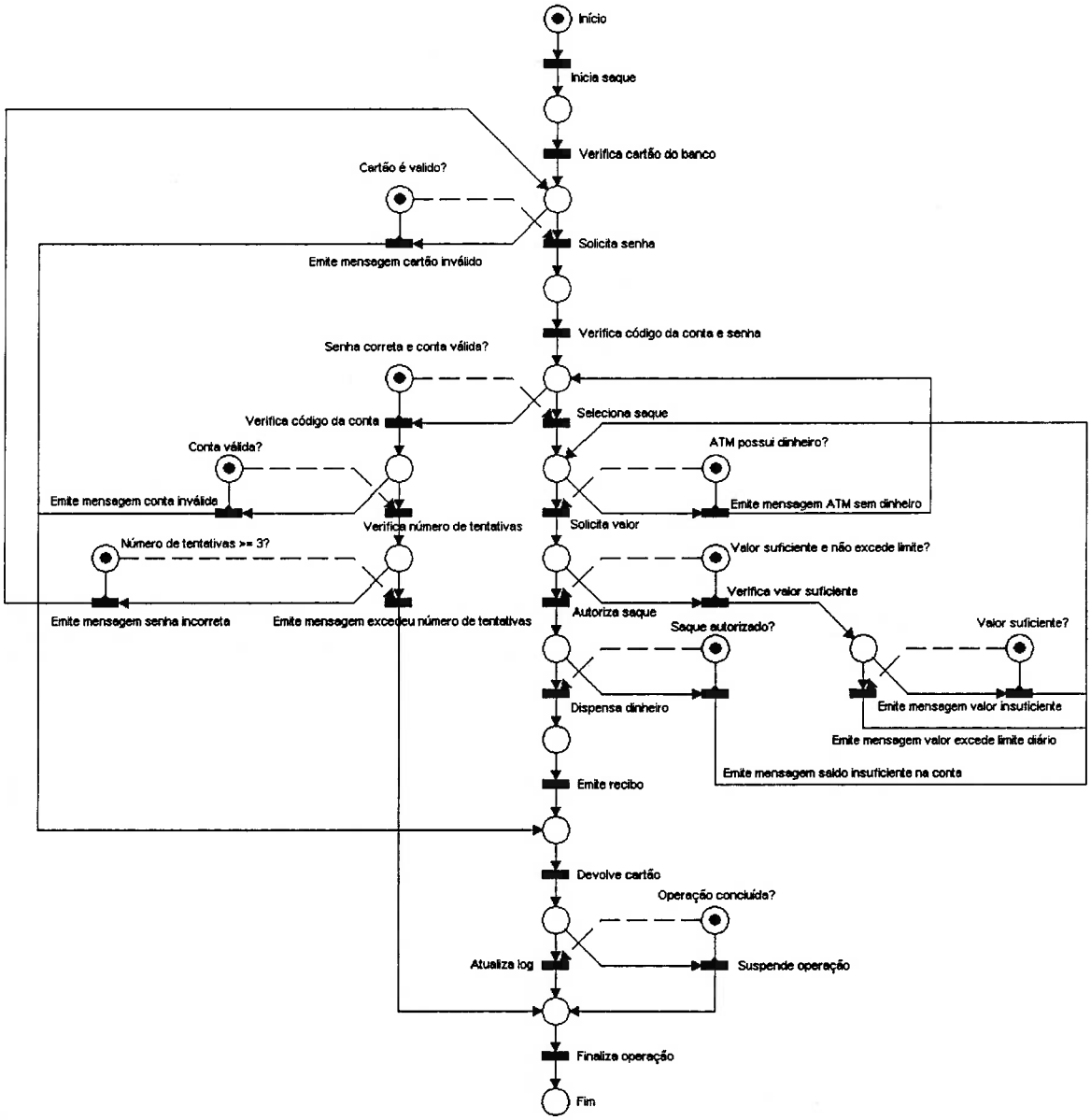


Figura 3-4 Rede de Petri equivalente ao Use Case Saque de Dinheiro

A seguir serão apresentados os resultados da proposta apresentada neste capítulo.

4 RESULTADOS

Como principal resultado do trabalho, o autor apresenta a seguir uma breve especificação de uma ferramenta que pode servir como apoio para a atividade de Verificação de Requisitos de Sistemas.

4.1 Especificação da Ferramenta de Verificação de Requisitos

Essa possível ferramenta possuirá como entrada a descrição do fluxo de eventos do Use Case.

Como principais funcionalidades, a ferramenta permitirá ao analista:

1. Classificar os eventos da descrição do Use Case conforme tipo de evento que consta da tabela 3-1 e atribuindo títulos aos mesmos, permitindo uma visualização em blocos, conforme descrição apresentada no item 3.2.4.1, sendo possível definir o nível que se deseja analisar.
2. Gerar uma rede de Petri equivalente à descrição do Use Case apresentado na tela para posterior verificação, segundo uma simulação que poderá executada em uma ferramenta que suporte tal característica.

3. Retornar as possíveis alterações efetuadas no modelo em rede de Petri para a descrição do Use Case.

Esta ferramenta deverá utilizar XML para a transferência de informação com os sistemas que a mesma possuir interface.

O autor acredita que a ferramenta aqui descrita, que arbitrariamente será chamada PN-RAVaT (Petri Net-Requirement Analysis and Validation Tool), deverá automatizar o processo deixando ao usuário apenas as decisões de alto nível.

Assim, o primeiro passo seria transformar um Use Case descrito em linguagem natural com identificadores numéricos (administrados pelo sistema) em um "Markup Use Case", como mostrado no Capítulo 3. Este pode ser transformado diretamente em uma rede de Petri Elementar (descrita no Capítulo 2) por mapeamento direto da estrutura sintática, gerando ao mesmo tempo a matriz de incidência desta rede.

A geração da matriz de incidência pode ser feita fazendo com que as sub-redes sejam próprias (o que é uma condição para a geração dos requisitos) e que podem ser acopladas em uma mesma matriz seguindo o método proposto por (Gonzalez del Foyo, 2001).

A verificação da convergência, feita neste trabalho de maneira informal, pode ser melhorada introduzindo a noção de isomorfismo entre redes, de modo a poder comparar o *viewpoint* de cada um dos atores considerados importantes: o Patrocinador, o Usuário final e o Desenvolvedor. Para facilitar esta tarefa, um analisador de redes e editor gráfico seria acoplado a esta ferramenta, embora acredite-se que o Usuário final vá preferir uma abordagem associada a um sistema de diagnóstico.

Neste ponto as idéias de (Leite, 1991) estão de pleno acordo com o presente trabalho, onde se prevê que a validação passe por analisar o impacto dos requisitos sobre uma base de conhecimento baseada na noção de contexto.

A figura 4-1 mostra a arquitetura do sistema proposto.

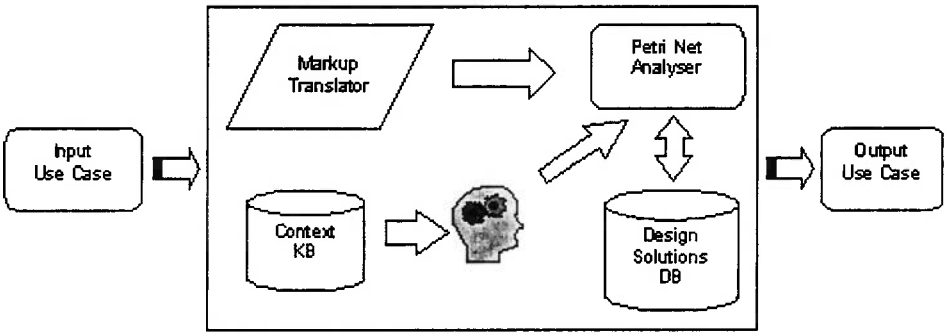


Figura 4-1 Arquitetura da ferramenta PN-RAVaT

5 DISCUSSÃO

O autor apresenta neste capítulo uma discussão sobre o alcance dos objetivos definidos anteriormente, assim como as limitações observadas no emprego da proposta descrita neste trabalho. Para que essas limitações sejam sanadas, e no sentido de proporcionar mais maturidade à proposta apresentada, serão discutidas algumas pesquisas a serem desenvolvidas com o enfoque de garantir a continuidade deste trabalho.

5.1 Objetivos Atingidos

O principal objetivo definido pelo autor foi o de contribuir para a redução da taxa de erros ocasionados na fase de especificação de requisitos de sistemas. Esse objetivo foi alcançado uma vez que a proposta permite ao analista interagir com os atores do sistema, verificando se a especificação que foi elaborada pelo analista vai de encontro com o que o usuário do sistema acredito ter expressado. Vale ressaltar que a abordagem utilizada, em considerar pelo menos os pontos de vistas do sistemas vistos por seus atores, conforme ilustrado na figura 2-2, tende a reduzir os enganos cometidos em se atender bem um grupo de usuários e, no entanto, deixar de cobrir as necessidades do outro grupo. Além deste aspecto, a verificação da especificação se dá utilizando-se de um formalismo matemático já bem difundido: a rede de Petri. Estes aspectos garantem à proposta uma base sólida para o seu propósito.

Um fato importante a ser citado, ainda em relação à utilidade da proposta, se deve ao interesse no trabalho que fora despertado na empresa fabricante de software, a Rational Software Corporation do Brasil. A mesma demonstrou interesse na pesquisa no sentido de considerar a agregação de uma possível ferramenta gerada a partir deste trabalho como um "add-on" a sua família de produtos, a Rational Suite.

5.2 Limitações da Proposta

Uma limitação importante observada pelo autor foi a falta de um método que possibilite a integração dos pontos de vistas observados pelos diferentes atores do sistema.

Outros fatores, que no momento podem causar um certo desconforto quando da utilização da proposta do trabalho, são a inexistência de uma interface que possibilite ao analista especificar os Use Cases seguindo a idéia do autor e a falta de uma ferramenta que possibilite ao usuário visualizar interativamente o modelo em rede de Petri do respectivo Use Case em questão.

5.3 Tendências das Ferramentas de Suporte a Engenharia de Requisitos

A literatura especializada em Engenharia de Requisitos mostra um crescimento do esforço acadêmico, e também de mercado, no sentido de prover métodos para uma efetiva eliciação de requisitos antes de se passar para as fases mais formais do desenvolvimento de sistemas. Registre-se que para sistemas automatizados, dotados de autonomia de ação (mesmo que restrita), a determinação precisa das funcionalidades das interações possíveis destes sistemas com os agentes que o cercam é fundamental, bem como o estabelecimento de fluxos alternativos e recuperação de situações de erro, bloqueios, etc.

Entretanto, se por um lado há uma evolução nos métodos de eliciação, não há o correspondente crescimento em ferramentas de apoio ao desenvolvimento de sistemas que suportem estas fases iniciais do processo (requisitos e especificações).

Por exemplo, com relação aos requisitos, o índice de ferramentas CASE da Universidade de Queens's (Queens's University, 2002) traz uma relação de cerca de 550 diferentes ferramentas destinadas ao suporte do ciclo de vida de software (onde há uma maior profusão de ferramentas de apoio). Desta lista, menos de 2,3 % (apenas 13, conforme mostra a tabela 5-1) são destinadas a apoiar a Engenharia de Requisitos. Ainda assim, quase todas as ferramentas listadas se dedicam à análise e gerenciamento de requisitos ou teste, associando código implementado e requisitos (e portanto não se destinam especificamente à fase inicial do desenvolvimento). Apenas uma ferramenta menciona a validação.

Ferramenta	Fabricante
Analyst Pro	Goda Software Inc.
Business Analyst/2020	PowerPlus System Inc.
Caliber RM	Starbase Co.
Caliber RBT	Technology Builders Inc.
Caliber RM	Technology Builders Inc.
Clyder	Sema Group
Cradle	3SL
DOORS	Telelogic AB
RDD-100	Ascent Logic Co.
RDT	Igatech Systems Pty Ltd.
SoftTest	BDQ Business Solutions
SoftTest	Bender and Associates
Xtie-RT	Teledyne Brown Engineering

Tabela 5-1 Ferramentas de apoio a Engenharia de Requisitos

A opinião do autor é que o trabalho de criação e exploração de novos métodos para Engenharia de Requisitos deveria privilegiar, na medida do possível, também o surgimento de novas ferramentas de eliciação, análise, gestão e validação de requisitos, associando teoria e prática.

A maioria dos métodos de eliciação de requisitos, como mostrado no Capítulo 2 enfoca principalmente a eliciação e análise. Somente um método, o VORV (Leite, 1991), se preocupa com a validação, ou melhor, em integrar a validação dos

requisitos como parte do processo de eliciação e análise, uma vez que o processo pressupõe a geração de requisitos corretos para o sistema como um todo e não apenas gerar requisitos simplesmente.

A opinião do autor é que um método consistente de análise de requisitos deverá surgir da fusão entre o VOSE, VORD e VORV, com ênfase neste último. Entretanto é preciso introduzir neste esforço uma certa preocupação com a geração de ferramentas CASE para dar suporte ao processo de desenvolvimento.

Exatamente por pensar desta forma, o autor preferiu não avançar muito na proposta de um novo método ou da fusão mencionada entre os métodos citados sem antes conseguir juntar ao esforço inicial de análise e validação a perspectiva de geração de uma ferramenta que tornasse o processo exeqüível. Tal ferramenta, apresentada de forma muito sucinta no Capítulo 4 está agora em discussão com a Rational Software Corporation, para teste e uma eventual inclusão na lista de ferramentas do Rational Suite.

Cuidados semelhantes podem ser notados na discussão iniciada no Capítulo 2, onde mencionou-se um contexto mais geral para trabalhar com a validação de requisitos. Pode-se notar que ao pensar no processo como um todo o autor está sempre direcionando os esforços para a convergência do processo, imaginando atores de destaque que possam fazer encerrar a comparação de *viewpoints*, como proposto em (Leite, 1991). Estas idéias deverão ser alvo de um trabalho futuro ao qual se faz referência no capítulo seguinte.

6 CONCLUSÃO

A possibilidade que a ferramenta de Verificação de Requisitos terá em se tornar tendência para Validação de Requisitos será grande, uma vez que a incorporação da mesma ao "Rational Suite" deverá despertar o interesse de outros fabricantes de ferramentas de apoio a Engenharia de Requisitos. Além disso, com o desenvolvimento dos trabalhos observados no item seguinte, a proposta deste trabalho pode se tornar um método para a Engenharia de Requisitos.

6.1 Trabalhos Futuros

O autor acredita que os trabalhos que deverão ser desenvolvidos futuramente estarão agregando valor à proposta apresentada. São eles:

1. Solidificação da fusão dos diversos pontos de vista do sistema, conforme proposto por (Leite, 1991), de forma a permitir que a verificação dos Requisitos do sistema seja mais intuitiva para os atores envolvidos no sistema. Esta seria a solução para os problemas hoje encontrados com a Especificação de Sistemas.

2. A adequação da proposta de uma rede de Petri Hierárquica extendida (Gonzalez del Foyo, 2001), para permitir a abstração das especificações

utilizando a capacidade de representar hierarquia dos pontos de vistas do sistema, através das sub-redes desta rede de Petri.

3. Estabelecimento de uma arquitetura de software para o desenvolvimento da Ferramenta de Verificação de Requisitos, onde toda a troca de informação, tanto com as ferramentas de especificação de requisitos, como com as ferramentas de edição e verificação de rede de Petri, serão baseadas em XML.

4. Propor uma alteração no diagrama de Atividades da UML, de forma que o mesmo possa ser passível de verificação, assim como os diagramas em rede de Petri.

Finalmente, com o apoio da técnica proposta, a possibilidade de se especificar sistemas de grande porte aumenta, uma vez que a divisão do mesmo em pequenas partes e a validação das mesmas pode ser feita recursivamente.

LISTA DE REFERÊNCIAS

Anschuetz, H. *A tool for design and simulation of Petri nets: HPSim version 1.1*. s.l.: s.ed., 2001. Disponível em: <http://home.t-online.de/home/henryk.a/petrinet/e/sim_form.htm>. Acesso em: 13 jul. 2002.

Booch, G. *Object oriented and design with applications*. Redwood City, Cal.: Benjamin/Cummnins, 1991.

Finkelstein, A.; Kramer, J.; Nuseibeh, B.; Goedicke, M. *Viewpoints: a framework for integrating multiple perspectives in systems development*. International Journal of Software Engineering and Knowledge Engineering, v.2, 1992.

González Del Foyo, P. M. *GHENeSys: uma rede estendida orientada a objetos para o projeto de sistemas discretos*. 2001. 125p. + apêndice. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2001.

Ghezzi, C.; Jazayeri, M.; Mandrioli, D. *Fundamentals of software engineering*. Englewood Cliffs: Prentice Hall, 1991.

Jacobson, I. *Object oriented software engineering: a use case driven approach*. New York: ACM Press, 1992.

Jacobson, I.; Booch, G.; Rumbaugh, J. *The unified software development process*. Reading, Mass: Addison–Wesley, 1999.

Kotonya, G.; Sommerville, I. *Requirements engineering with viewpoints*. Software Engineering Journal, v.11, n.1, p.5-18, jan., 1996.

Kotonya, G., Sommerville, I. *Requirements engineering: processes and techniques*. Chichester: John Wiley, 1998.

Leite, J.C.P.; Freeman, P.A. *Requirements validation through viewpoint resolution*. IEEE Transactions Software Engineering, v.17, n.12, p. 1253-1269, dez., 1991

Marca, D. *SADT: structured analysis and design technique*. New York: McGraw Hill, 1988.

Mullery, G. *A method for controlled requirements specifications*. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 4., Munich, Germany. Proceedings. New York: Institute of Electrical and Electronics Engineers, 1979.

Murata, T. *Petri Nets: properties, analysis and applications*. Proceedings of the IEEE, v.77, n. 4, p. 541-580, abr., 1989.

Naur, Peter (ed.) *Revised report on the algorithmic language ALGOL 60*. Communications of the ACM, v. 3, n.5, p. 299-314, 1960.

Object Management Group. *Unified modeling language specification: version 1.4*. s.l.: s.ed., 2001. Disponível em: <http://www.omg.org/uml>. Acesso em: 16 mar. 2002.

Pressman, R. S. *Software engineering: a practitioner's approach*. New York: McGraw–Hill, 1993.

Queens's University. School of Computing. *Case tool index*. Kingston: s.ed., 2002. Disponível em: <http://ww.qucis.queensu.ca/Software-Engineering/tools.html>. Acesso em: 9 nov. 2002.

Ross, D. *Applications and extensions of SADT.* IEEE Computer, v.18, n.4, p. 25-34, 1985.

Rumbaugh, J., et al. *Object oriented modeling and design.* Englewood Cliffs: Prentice Hall, USA, 1991.

Rational Software Corporation. *Rational unified process: version 2001A.04.00.* s.l.: s.ed., 2001.

Sanders, B.W. *A software engineer's guide to the galaxy*: unpublished notes for senior software engineering project. Colorado: University of Colorado at Boulder Department of Computer Science, 1987-2001. Disponível em: http://www.cs.colorado.edu/~sanders/cs4308/class/guide/lifecycle/lifecycle.html>. Acesso em: 20 jul. 2002.

Silva, J.R.; Miyagi, P.E. *A formal approach to PFS/MFG: a Petri net representation of discrete manufacturing systems.* Studies in Informatics and Control, v.5, n.2, p.131-141, jun., 1996.

Silva, J.R. *Interactive design of integrated systems.* In: IEEE/IFIP INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY FOR BALANCED AUTOMATION SYSTEMS IN MANUFACTURING, 3., Prague, Czech Republic, 1998. Intelligent systems for manufacturing: proceedings. Boston: Kluwer Academic Publishers, p. 567-578, 1998.

Sommerville, I.; Sawyer, P. *Requirements engineering: a good practice guide.* New York: John Wiley & Sons, 1997.

Vernadat, F. *Enterprise modeling and integration: principles and applications.* London: Chapman & Hall, 1996.

Yourdon, E. *Modern structured analysis.* Englewood Cliffs: Yourdon Press, 1989.