

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

PEDRO BARBOSA CORDEIRO

**PROJETO E IMPLEMENTAÇÃO DO MÓDULO
TAME DA FERRAMENTA ITAOS PARA
ANÁLISE E MODELAGEM DA TAREFA**

Campina Grande - PB

Fevereiro de 2003

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

PEDRO BARBOSA CORDEIRO

PROJETO E IMPLEMENTAÇÃO DO MÓDULO
TAME DA FERRAMENTA ITAOS PARA
ANÁLISE E MODELAGEM DA TAREFA

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (MSc).

Instituição: Universidade Federal de Campina Grande

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Engenharia de Software

Orientador: Bernardo Lula Jr., Dr.

Campina Grande - PB

Fevereiro de 2003

CORDEIRO, Pedro Barbosa

C794P

Projeto e Implementação do Módulo TAME da Ferramenta iTAOS para Análise e Modelagem da Tarefa.

Dissertação (Mestrado), Universidade Federal de Campina Grande, Coordenação de Pós-Graduação em Informática, Campina Grande – Paraíba, Fevereiro de 2003.

116 p. Il.

Orientador: Bernardo Lula Júnior

Palavras-Chave:

1. Tarefa
2. Análise e Modelagem da Tarefa
3. Interfaces Homem-Máquina
4. Engenharia de Software

CDU – 519.683


**“PROJETO E IMPLEMENTAÇÃO DO MÓDULO TAME DA FERRAMENTA
ITAOS PARA ANÁLISE E MODELAGEM DA TAREFA”**

PEDRO BARBOSA CORDEIRO

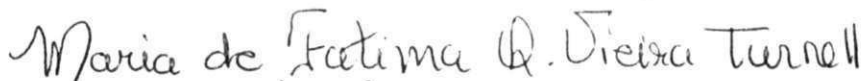
DISSERTAÇÃO APROVADA EM 26.02.2003



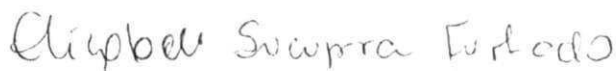
**PROF. BERNARDO LULA JÚNIOR, Dr.
Orientador**



**PROFª FRANCILENE PROCÓPIO GARCIA, D.Sc
Examinadora**



**PROFª MARIA DE FÁTIMA Q. V. TURNELL, Ph.D
Examinadora**



**PROFª MARIA ELIZABETH SUCUPIRA FURTADO, Drª
Examinadora**

CAMPINA GRANDE – PB

*Dedico este trabalho à
minha família e a meus amigos.*

AGRADECIMENTOS

Primeiramente, agradeço a Deus, por ter permitido que eu chegasse até aqui e ter concluído este trabalho.

À minha mãe Vera Lúcia Barbosa Cordeiro, pelo incentivo e apoio durante todo esse período e pela constante preocupação com a minha formação.

Ao meu professor orientador Bernardo Lula Jr., por toda a sua dedicação, apoio, e discussões na realização deste trabalho.

A meu colega Francisco Petrônio Alencar de Medeiros, por ter cooperado para o andamento deste trabalho.

Sumário

Lista de Figuras	ix
Lista de Tabelas	x
Lista de Descritores.....	xi
Resumo.....	xii
Abstract	xiii
Capítulo 1 – Introdução	1
1. PROJETO DE INTERFACES HOMEM-MÁQUINA	1
2. FORMALISMOS PARA A ANÁLISE DA TAREFA.....	2
3. ÍTAOS.....	4
4. OBJETIVO.....	5
5. HIPÓTESES	6
6. METODOLOGIA	6
7. ORGANIZAÇÃO DO TRABALHO.....	7
Capítulo 2 – Ferramentas para Análise e Modelagem da Tarefa	9
1. INTRODUÇÃO	9
2. CARACTERÍSTICAS	9
2.1. <i>Suporte a Construção de Modelos</i>	10
2.2. <i>Métricas para Modelos de Tarefas</i>	10
2.3. <i>Simulação</i>	11
2.4. <i>Geração de Interfaces do Usuário</i>	11
3. COMO ESTÃO AS FERRAMENTAS ATUALMENTE?.....	11
4. CTTE	12
5. EUTERPE.....	13
6. IMAD*.....	14
7. FUNCIONALIDADES DE ÍTAOS.....	14
8. CONCLUSÃO	15
Capítulo 3 – A Linguagem TAOS	16
1. INTRODUÇÃO	16
2. META-MODELO TAOS.....	16
3. CONCEITOS ESTÁTICOS.....	17
4. CONCEITOS DINÂMICOS.....	19
5. EXTENSÃO DE TAOS	21
6. EXEMPLO	23
7. CONCLUSÃO	26
Capítulo 4 – O Processo de Desenvolvimento	27
1. INTRODUÇÃO	27
2. ESCOLHA DO PROCESSO.....	27
3. O PROCESSO UNIFICADO	28
3.1. <i>Características do Processo Unificado</i>	29
3.2. <i>Ciclos de Vida</i>	30

3.3. <i>Fases de um Ciclo de Vida</i>	30
3.4. <i>Iterações</i>	31
3.5. <i>Fluxos de Atividades (Workflows)</i>	31
3.6. <i>Atividades</i>	32
4. A ADAPTAÇÃO.....	33
5. TAREFAS VS. <i>USE CASES</i>	34
5.1. <i>Use Cases</i>	35
5.2. <i>Semelhanças</i>	35
5.3. <i>Vantagens e Desvantagens</i>	36
6. DESENVOLVIMENTO DO MÓDULO TAME.....	37
6.1. <i>Fase de Planejamento</i>	37
5.2. <i>Fase de Elaboração</i>	39
5.3. <i>Fase de Construção</i>	40
5.4. <i>Fase de Transição</i>	41
6. CONCLUSÃO	42
Capítulo 5 – Projeto Arquitetural.....	43
1. INTRODUÇÃO	43
2. ESTRUTURA DO SISTEMA	43
2.1. <i>Interfaces de Classes</i>	43
2.2. <i>Classes Abstratas</i>	44
2.3. <i>Métodos e Atributos</i>	45
3. COMPORTAMENTO DINÂMICO DO SISTEMA	45
4. PERSISTÊNCIA DOS MODELOS DE TAREFA	46
4.1. <i>JATO</i>	46
4.2. <i>Classes de Persistência</i>	47
5. PADRÕES DE PROJETO.....	48
5.1. <i>Façade</i>	48
5.2. <i>Abstract Factory</i>	48
5.3. <i>Factory Method</i>	49
5.4. <i>Singleton</i>	49
5.5. <i>Observer</i>	49
5.6. <i>Template Method</i>	51
5.7. <i>Interpreter</i>	51
6. CONCLUSÃO	52
Capítulo 6 – Descrição do Sistema.....	53
1. INTRODUÇÃO	53
2. IDENTIFICAÇÃO DOS REQUISITOS FUNCIONAIS	53
3. CRIAÇÃO DE NOVOS MODELOS DE TAREFAS	54
4. SALVAR MODELOS DE TAREFAS	54
5. RECUPERAR MODELOS DE TAREFAS EXISTENTES	54
6. EDIÇÃO DE MODELOS DE TAREFAS.....	54
7. REUSO DE CONCEITOS	55
8. ADIÇÃO DE ATRIBUTOS AOS CONCEITOS.	55
9. IMPRESSÃO DE MODELOS DE TAREFAS	56
10. VERIFICAÇÃO DA COMPLETUDE DAS INFORMAÇÕES.....	56
11. VERIFICAÇÃO DA COERÊNCIA DAS INFORMAÇÕES.....	56
11.1. <i>Regras para Verificação das Situações</i>	56
12. SIMULAÇÃO DE MODELOS DE TAREFAS	57

13. CONCLUSÃO	58
Capítulo 7 – Testes Funcionais	59
1. INTRODUÇÃO	59
2. PLANEJAMENTO DOS TESTES	59
2.1. <i>O Que Testar</i>	59
2.2. <i>Recursos Utilizados</i>	60
3. PROJETO DOS TESTES	60
3.1. <i>Casos de Teste</i>	61
4. IMPLEMENTAÇÃO DOS TESTES	63
5. EXECUÇÃO DOS TESTES	63
5.1. <i>Resultados</i>	64
6. CONCLUSÃO	64
Capítulo 8 – Discussões e Conclusão	65
1. INTRODUÇÃO	65
2. DISCUSSÕES DOS RESULTADOS	65
3. CONTRIBUIÇÕES	67
4. TRABALHOS FUTUROS	67
9. Referências Bibliográficas	69
Anexo A: Documento de Visão	73
Anexo B: Plano de Gerenciamento de Requisitos	81
Anexo C: Glossário	88
Anexo D: DTD (Document Type Definition)	95
Anexo E: Resultados dos Testes	98

Lista de Figuras

Figura 1: Princípio da Independência do Diálogo	5
Figura 2: CTTE.....	12
Figura 3: EUTERPE	13
Figura 4: Meta-Modelo da Linguagem TAOS em UML	17
Figura 5: Meta Modelo da Linguagem TAOS Estendida.....	21
Figura 6: Extrato da tarefa “Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas”	23
Figura 7: Blocos chave do Processo Unificado	29
Figura 8: As quatro fases do Processo Unificado.....	30
Figura 9: Fases e <i>workflows</i> do Processo Unificado	32
Figura 10: Arquitetura base do módulo TAME na notação UML	38
Figura 11: Relacionamento inter-classes da arquitetura base do módulo TAME.....	39
Figura 12: Estrutura de pacotes do módulo TAME.....	39
Figura 13: Hierarquia de interfaces da arquitetura base	44
Figura 14: Hierarquia de classes abstratas da arquitetura base	45
Figura 15: Classe responsável pela persistência dos modelos.....	47
Figura 16: Padrão Observer	50
Figura 17: Classes que representam os operadores da linguagem TAOS	51
Figura 18: Estrutura de Tarefas	57

Lista de Tabelas

Tabela 1: Matriz de Comparação das Ferramentas	14
Tabela 2: Atividades do PU e seus principais fluxos de atividades	33
Tabela 3: Fluxo de Atividades do Processo do Módulo TAME.....	34
Tabela 4: Casos de Teste	62
Tabela 5: Casos de Teste (Continuação)	63

Lista de Descritores

Descritor 1: Descritores da tarefa raiz	24
Descritor 2: Descritores da pré-situação da tarefa raiz.....	25
Descritor 3: Descritores do método da tarefa raiz	25
Descritor 4: Descritores da ação: Ativar Ambiente de Modelagem.....	25
Descritor 5: Descritores do agente: Projetista de Interface	26
Descritor 6: Descritores da ferramenta: Mouse	26

Resumo

CORDEIRO, P. B. *Projeto e Implementação do Módulo TAME da Ferramenta iTAOS para Análise e Modelagem da Tarefa*. Dissertação (Mestrado em Informática) – Coordenação de Pós-Graduação em Informática. Campina Grande: Universidade Federal de Campina Grande, 2003.

A demanda por *softwares* interativos com alto grau de usabilidade é crescente. Nota-se cada vez mais a importância desse tipo de software, tanto para o usuário como para a empresa que o fabrica. Para que seja possível alcançar o grau de usabilidade adequado, se faz necessário fazer um projeto do sistema centrado no usuário, levando-se em consideração a interface do usuário desde as primeiras fases do processo de desenvolvimento do sistema. Embora as metodologias tradicionais de desenvolvimento de software só considerem o projeto de interface como um apêndice ao processo de desenvolvimento, o campo de interface homem-máquina vem tomando relevo como um campo próprio no processo de desenvolvimento de software e produzindo suas próprias metodologias. Nesse sentido, destacam-se hoje as metodologias que fazem uso da análise e modelagem da tarefa para incorporar o conhecimento do usuário sobre sua tarefa no projeto de interface. Porém, essa técnica carece de suporte computacional adequado, maduro o suficiente e disponível para que ela seja praticável. Em vista disso, esse trabalho tem como objetivo implementar e projetar o módulo funcional de uma ferramenta dessa categoria. O módulo em questão é o módulo TAME, que implementa os requisitos funcionais levantados para a ferramenta iTAOS para análise e modelagem da tarefa.

Abstract

CORDEIRO, P. B. *Design and Implementation of iTAOS' TAME Module for Task Analysis and Modeling [Projeto e Implementação do Módulo TAME da Ferramenta iTAOS para Análise e Modelagem da Tarefa]*. Dissertação (Mestrado em Informática) – Coordenação de Pós-Graduação em Informática. Campina Grande: Universidade Federal de Campina Grande, 2003.

The demand for interactive software with a high degree of usability has been increasing. The importance of such kind of software is highly noted, it is important for the user and for the company that produces it. To obtain an adequate degree of usability, it is necessary to make a design centered on the user, considering its interface of interaction since the firsts phases of its development process. The traditional software development methodologies consider the user interface as an appendix of the system, due to this, the field of human computer interface has done researches to develop its own methodologies. Towards this approach, nowadays some methodologies based in the task analysis and modeling to incorporate the user knowledge about its task within the design of the interface worth distinction. Nevertheless this technique has a lack of computational support. This work has the goal of design and construct a functional component to be used in the construction of a computational tool, iTAOS tool, to give support to task analysis and modeling. This functional component is the TAME module, that implements the functional requirements raised to iTAOS tool.

Capítulo 1 – Introdução

1. Projeto de Interfaces Homem-Máquina

Segundo Shneiderman (Shneiderman, 1998), muitos projetos de *software* falham em atingir seus objetivos, ou seja, não terminam dentro do prazo, gastam além do orçamento estimado e/ou não satisfazem aos requisitos funcionais. Algumas estimativas sugerem que esse número pode ser superior a 60%, dos quais, cerca de 25% dos projetos nunca são terminados e 35% atingem seus objetivos parcialmente. Muitos desses problemas podem ser atribuídos à falta de um projeto de interface do usuário desde os primeiros estágios do processo de desenvolvimento.

Se a interface do usuário for levada em conta desde os primeiros estágios do processo de um *software*, seu tempo de desenvolvimento e seu custo tendem a cair drasticamente. Sistemas com uma interface do usuário bem projetada custam menos para desenvolver, pois o tempo de desenvolvimento gasto com a implementação e, posteriormente, com re-trabalho, tende a cair. A manutenção de tais sistemas durante seu ciclo de vida é mais barata, já que chamadas à equipe de suporte por razões de utilização do sistema serão mais escassas, considerando que uma interface bem projetada é mais fácil de ser utilizada. Tais sistemas também seriam mais fáceis de aprender, agilizariam a atividade do usuário e reduziriam suas chances de cometer erros de operação, ou seja, apresentariam um maior grau de usabilidade.

O mercado corporativo e os departamentos de assistência ao consumidor estão cada vez mais cientes da importância da usabilidade em sistemas computacionais. Quando empresas concorrentes lançam no mercado um produto com funcionalidades similares, a usabilidade é um fator vital para a aceitação do produto.

Sendo assim, desenvolvedores de *software* que não se preocupam em fazer um bom projeto da interface do usuário de seus sistemas podem perder clientes para seus concorrentes, ainda mais com a facilidade que estes encontram hoje para testar versões de demonstração de diversos fabricantes sem custos adicionais ou compromisso. Obviamente o consumidor vai preferir comprar aquele *software* com o qual teve mais familiaridade, ou seja, que achou mais fácil de usar.

Não se atinge a usabilidade de um sistema computacional interativo sem um projeto de interface do usuário, a ser considerado desde o princípio no processo de desenvolvimento do *software*. A questão principal no projeto de *software* interativo, para atender os requisitos de usabilidade, é tornar as representações propostas para a interface compatíveis com aquelas desenvolvidas pelos usuários em seu trabalho.

Em vista de produzir sistemas computacionais com interfaces do usuário (IU) que proporcionem as qualidades descritas acima e que englobem as funcionalidades que o sistema deve oferecer, empresas produtoras de *software* vêm investindo cada vez mais no projeto da IU de seus sistemas. Para que seja possível desenvolver tais tipos de interfaces, se faz necessário o conhecimento dos futuros usuários do sistema a ser desenvolvido e do seu contexto de utilização, ou seja, deve ser feito um projeto centrado no usuário e na tarefa a ser realizada pelo mesmo (UCD – *User Centered Design*).

2. Formalismos para a Análise da Tarefa

A análise da tarefa é considerada fundamental não apenas para assegurar um projeto centrado no usuário, como também para melhorar o entendimento de como o usuário deve interagir com a interface do sistema para executar sua tarefa (Limbourg, Pribeanu e Vanderdonckt, 2001). Com a análise da tarefa busca-se uma descrição da tarefa em termos de objetivos, procedimentos, restrições, etc.

A análise da tarefa (AT) é um princípio fundamental da Ergonomia, que é o conhecimento do usuário e do trabalho a ser realizado pelo mesmo (Cibys, 1996; Sebillote, 1995). A AT é realizada, principalmente, com entrevistas dirigidas a gerentes e usuários e com observações do trabalho realizado, buscando evidenciar tanto a lógica de funcionamento como a lógica de utilização de um sistema. A lógica de funcionamento é aquela desenvolvida através do conhecimento dos aspectos internos de funcionamento do sistema e baseia-se no conhecimento das funções e de seus mecanismos internos. A lógica de utilização é desenvolvida através do conhecimento da interação com o sistema e é baseada na chamada imagem operativa, que é a representação que se tem da realidade do ambiente de trabalho, modificada e simplificada pelo que é funcionalmente significativo. A imagem operativa amplia os elementos pertinentes e elimina os secundários (Falzon, 1989).

Quando uma perspectiva para o projeto de interface baseado no usuário é adotada, ergonomistas, que são especialistas em análise da tarefa, produzem um documento na forma

de texto, que contém informações a respeito dos requisitos do usuário e de sua tarefa. Nas mãos dos projetistas de interfaces, geralmente pessoas do campo da Informática, especializados na área de Interfaces Homem-Máquina (IHC), tais requisitos são traduzidos em especificações de interface de alto nível e logo depois em especificações de interface propriamente ditas. Geralmente esse processo é executado de uma maneira informal, pois a cooperação entre os ergonomistas e os projetistas se dá por meio da troca de requisitos. E o problema está no fato de que os ergonomistas geralmente utilizam maquetes e rascunhos para representar tais requisitos, enquanto que os projetistas utilizam modelos, notações e protótipos.

Em vista de melhorar a comunicação entre ergonomistas e projetistas de interfaces, alguns formalismos para representação de tarefa surgiram como resultados de trabalhos em Ergonomia Cognitiva junto com a contribuição muito importante da Inteligência Artificial (IA). Entre tais formalismos podemos citar: MAD (Scapin, Pierret e Christine, 1989), MAD* (Hammouche, 1995), TKS (Johnson, P., Johnson, H., Waddington *et al.*, 1988) e TAOS (Medeiros e Rousselot, 1995, Medeiros, 1998a, Medeiros, 1998b, Medeiros, Kafure e Lula, 2000). Todos esses formalismos de descrição de tarefas assumem que o conhecimento que uma pessoa tem a respeito de sua tarefa é hierarquicamente estruturado de acordo com o paradigma da planificação hierárquica (PH) (Sacerdoti, 1974). Eles também manipulam basicamente os mesmos conceitos, tais como: tarefas, ações, objetivos, etc. As estruturas formais utilizadas para descrever os conceitos são baseadas no conceito de *frame*, também oriundo da IA (Minsky, 1975).

Vimos que a análise da tarefa pode ter suporte de formalismos, mas o uso de formalismos não resolve o problema relacionado com o consumo de tempo que essa atividade requer e nem a torna mais fácil de ser realizada. Portanto, certamente as pessoas que realizam a análise da tarefa têm muito a ganhar com o suporte computacional, que pode reduzir os esforços expendidos durante sua realização. Uma ferramenta computacional pode também aumentar a qualidade da análise já que os modelos gerados são construídos com a ajuda de um guia, que é a própria ferramenta. E existe também um requisito muito forte, imposto pelo mercado, de que uma técnica de modelagem deve ser suportada por uma ferramenta, ou ela não será utilizada (Welie, 2001) Existem também alguns outros problemas relacionados com a análise e modelagem da tarefa sem um suporte computacional que serão abordados no próximo capítulo.

Algumas ferramentas já foram propostas e implementadas por grupos de pesquisa, entre elas podemos citar: CTTE (Paternò 1999), Euterpe (Welie, Veer e Eliëns, 1998a, Welie, Veer, Eliëns, 1998b) e IMAD (Gamboa, Scapin, 1997).

3. iTAOS

TAOS (Task and Action Oriented System) é um formalismo oriundo da IA, concebido para a aquisição e representação do conhecimento sobre um domínio, desenvolvido por J. H. de Medeiros (Medeiros e Rousselot, 1995, Medeiros, 1998a, Medeiros, 1998b, Medeiros, Kafure e Lula, 2000) e validado segundo MAD (Kafure, 2000) como formalismo para a descrição e análise de tarefas em vista da concepção de interfaces homem-computador de sistemas computacionais.

Trata-se de um formalismo centrado na análise das tarefas a serem executadas no domínio. Essa análise dá origem a uma representação estruturada dos conceitos estáticos (objetos, métodos e situações) e dinâmicos (processos, planos e ações que operam sobre as entidades estáticas) envolvidos no processo de realização da tarefa.

A linguagem TAOS vem sendo usada como linguagem de modelagem de tarefas do usuário pelo grupo de interfaces homem-máquina (GIHM) da Universidade Federal de Campina Grande (UFCG). Sendo a linguagem TAOS o objeto de estudo deste trabalho, teremos um capítulo dedicado à sua descrição, contendo informações a respeito de cada um dos conceitos definidos por ela.

TAOS foi concebido para ser implementado em dois módulos: TAME (Task and Action Modeling Environment) e TAOS-Graph.

O módulo TAME define a semântica da linguagem TAOS, que é uma linguagem semelhante a KL-ONE e permite a modelagem do conhecimento conceitual estático na forma de uma taxonomia e ainda faz melhoramentos aos descendentes convencionais de KL-ONE para incluir a descrição de mudanças dinâmicas de estado por meio de um formalismo provindo do campo das linguagens planificadas tais como: CLASP (Devambu e Litman, 1991), RAT (Heinsohn, Kudenko, Nebel *et al.*, 1992) e KRSL (Lehrer, 1993).

TAME permite modelar o conhecimento do domínio fazendo uma verificação desse conhecimento tanto do ponto de vista da completude (representação do conhecimento) como da validação da coerência desse conhecimento.

O módulo TAOS-Graph define a visualização gráfica do processo de modelagem definido por TAME. Trata-se de um editor de conceitos estáticos e dinâmicos que age de acordo com os princípios preconizados por TAME.

A junção desses dois módulos resulta em uma ferramenta de suporte a análise e modelagem da tarefa baseada no formalismo TAOS, a ferramenta iTAOS (Medeiros, Cordeiro e Lula, 2002b, Medeiros, Cordeiro e Lula, 2002c).

4. Objetivo

Dada a dificuldade de acesso a ferramentas externas maduras o suficiente para serem utilizadas pelo nosso grupo de interfaces homem-máquina e a grande importância de uma ferramenta do gênero, este trabalho tem como objetivo implementar o módulo TAME da ferramenta iTAOS. Ou seja será implementado o módulo funcional de uma ferramenta de suporte à análise e modelagem da tarefa baseada na linguagem TAOS.

A base para a funcionalidade da ferramenta foi extraída de funcionalidades de ferramentas já existentes. As características da ferramenta serão discutidas no próximo capítulo.

A ferramenta iTAOS é formada por dois módulos, sendo um módulo funcional e outro de interface, segundo o princípio da independência do diálogo, ilustrado na Figura 1 abaixo:

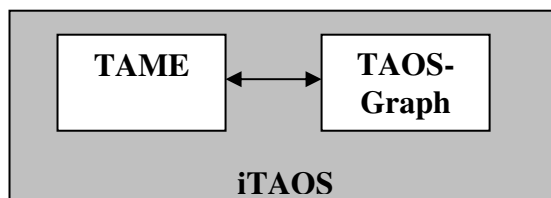


Figura 1: Princípio da Independência do Diálogo

Tal princípio diz respeito à separação modular entre o código da aplicação e o código da interface do usuário e trata-se de um princípio essencial que deve ser respeitado no processo de concepção de sistemas interativos (Ehrich e Hartson, 1981, Draper e Norman, 1985, Dodani, Hughes e Moshell, 1989).

5. Hipóteses

O trabalho será desenvolvido com base nas seguintes hipóteses:

- H1: É possível implementar a parte funcional e a interface do usuário de um sistema interativo em processos à parte, segundo o princípio da independência do diálogo.
- H2: É possível deduzir os requisitos funcionais de um sistema computacional a partir da análise e modelagem da tarefa do usuário para o qual o sistema se destina.
- H3: O Processo Unificado (PU) (Hunt, 2001, Larman, 2002) é um processo adequado para o desenvolvimento do módulo TAME.

6. Metodologia

A metodologia utilizada para o desenvolvimento do módulo TAME é a seguinte:

1. Levantamento de Requisitos
 - 1.1. Análise e modelagem iterativa e interativa da tarefa: “Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas” com o próprio formalismo TAOS, por meio de entrevistas com o usuário.
 - 1.2. Identificação de requisitos funcionais.
 - 1.3. Identificação de requisitos não-funcionais.
 - 1.4. Elaboração de um glossário.
 - 1.5. Elaboração de um documento de visão.
 - 1.6. Elaboração de um plano de gerenciamento de requisitos.
2. Análise
 - 2.1. Identificação dos objetos do domínio e suas classes.
 - 2.2. Identificação dos atributos das classes identificadas.
 - 2.3. Identificação dos relacionamentos entre as classes.
 - 2.4. Agrupamento das classes em pacotes.
3. Projeto
 - 3.1. Projeto da arquitetura base.
 - 3.2. Identificação de classes de projeto.
 - 3.3. Identificação do comportamento dinâmico das classes.

- 3.4. Geração de diagramas de classes e pacotes.
- 4. Implementação
 - 4.1. Implementação da arquitetura base.
 - 4.2. Definição de um modelo de implementação.
 - 4.3. Implementação do restante das classes.
- 5. Testes Funcionais
 - 5.1. Planejamento dos testes funcionais.
 - 5.2. Projeto dos testes funcionais.
 - 5.3. Execução dos testes funcionais.
- 6. Integração com o módulo TAOS-Graph.
- 7. Disponibilização da ferramenta iTAOS.

7. Organização do Trabalho

O trabalho está organizado em 8 capítulos. O Capítulo 1 contém as informações que são essenciais para um bom entendimento do que consiste o trabalho, tais como o objeto de estudo, o objetivo, as hipóteses consideradas e a metodologia usada para a implementação do módulo TAME.

O Capítulo 2 fornece um embasamento a respeito do que são ferramentas para análise e modelagem da tarefa, descrevendo as funcionalidades que elas devem oferecer e que ferramentas desse tipo estão disponíveis na atualidade.

O Capítulo 3 descreve o formalismo TAOS, dando ênfase ao seu meta-modelo (semântica).

O Capítulo 4 contém informações detalhadas a respeito do processo adotado para desenvolver o módulo TAME, mostrando passo a passo todas as fases do mesmo, bem como o fluxo de atividades, decisões tomadas e artefatos gerados durante os mesmos.

O Capítulo 5 descreve o projeto arquitetural do módulo TAME, mostrando as principais classes que o compõem, bibliotecas utilizadas para sua implementação e os principais Padrões de Projeto (*Design Patterns*) utilizados.

O Capítulo 6 descreve as funcionalidades identificadas durante a análise e modelagem da tarefa realizada no início do processo de desenvolvimento adotado.

O Capítulo 7 mostra o projeto de testes feito para verificar se todas as funcionalidades descritas no Capítulo 5 foram implementadas, e implementadas corretamente, bem como os resultados obtidos durante tais testes.

O Capítulo 8 mostra os resultados obtidos durante o processo de desenvolvimento do módulo TAME, contém discussões a respeito das hipóteses consideradas e algumas sugestões para trabalhos futuros.

Capítulo 2 – Ferramentas para Análise e Modelagem da Tarefa

1. Introdução

Esse capítulo descreve o estado atual das ferramentas de análise e modelagem de tarefas, enfatizando os principais requisitos que elas devem obedecer. Ao longo do mesmo são citadas algumas ferramentas existentes e comentadas algumas de suas principais características. E para finalizar, é feita uma comparação, em termos de funcionalidades, entre as ferramentas citadas e a ferramenta iTAOS.

Uma ferramenta dessa categoria oferece uma série de vantagens:

- Ambiente adequado para que ergonomistas e projetistas de interface possam utilizar o formalismo em questão de forma correta.
- Possível geração automática ou semi-automática da interface do usuário a partir do modelo da tarefa.

Além de tais vantagens, tal ferramenta pode solucionar alguns problemas que podem ser encontrados durante a atividade de modelagem de tarefas manualmente:

- O trabalho tedioso de descrever árvores de tarefas e notações formais com papel e caneta.
- Dificuldade de manipulação de modelos de tarefas armazenados em documentos de texto (*ie*: numerar as tarefas e suas sub-tarefas corretamente).
- Dificuldade com a verificação manual da coerência e completude da informação coletada.

2. Características

Algumas das características que podemos encontrar atualmente nas ferramentas existentes podem ser vistas nesta seção.

2.1. Suporte a Construção de Modelos

Uma funcionalidade básica para esse tipo de ferramenta é a possibilidade de construir modelos de tarefas. Existem muitas formas de se fazer isso, algumas permitem que os modelos sejam gerados a partir de *logs* de interação com sistemas. Porém, os modelos gerados com essa técnica têm uma grande limitação, pois refletem apenas o uso passado do sistema, e não apresenta nenhum potencial de uso futuro.

Outra maneira de gerar modelos de tarefas é a partir de análises textuais, onde a ferramenta associa automaticamente tarefas com verbos e objetos com nomes. Essa técnica produz alguns bons resultados, mas é muito simples para a obtenção de resultados mais gerais.

Algumas ferramentas oferecem também o suporte à importação de modelos do Rational Rose, uma ferramenta para modelagem de sistemas em UML. E existem algumas que têm a capacidade de receber uma descrição textual informal ou um *use case* e permitem que o projetista selecione interativamente as informações de interesse para o modelo.

Claro que em algumas dessas ferramentas o projetista não é obrigado a construir seu modelo das maneiras descritas acima, ele pode começar seu modelo do ponto zero e incrementá-lo à medida que ele vá conhecendo o domínio a ser modelado.

2.2. Métricas para Modelos de Tarefas

A estrutura e a informação que um modelo de tarefas pode conter proporciona informações úteis para o projetista. Algumas dessas informações podem ser usadas pelos projetistas para comparar modelos e saber como as pessoas trabalham no sistema atual e como trabalhariam em um outro sistema, ou mesmo para comparar duas alternativas de projeto. Atualmente não se conhece nenhuma ferramenta que ofereça esse tipo de métrica.

As métricas que algumas das ferramentas atuais obedecem estão relacionadas à quantidade de tarefas do modelo, à quantidade de tarefas básicas, ao número de instâncias de operadores temporais, à quantidade de níveis, etc. Com esse tipo de informação é possível comparar dois modelos separadamente e tomar algumas decisões de projeto. Por exemplo, um modelo com maior número de tarefas automáticas e um menor número de tarefas manuais leva a entender que é necessário alocar mais recursos para o sistema.

2.3. Simulação

Um simulador para modelos de tarefas pode ser útil para analisar melhor o comportamento dinâmico dos mesmos. Essa funcionalidade ganha ainda mais sentido quando a notação usada permite a especificação de relações temporais entre tarefas, tais como: concorrência, suspensão de tarefas ou mesmo interrupção. Poucas ferramentas oferecem essa funcionalidade. A simulação é uma ferramenta muito importante para que o analista possa entender o comportamento dinâmico de um sistema mais facilmente.

A simulação pode ser útil em diversos casos:

- Os projetistas podem verificar que o que estão modelando é o que realmente estão querendo representar. Isso é muito importante, principalmente no caso de grandes especificações, que são mais difíceis de entender devido à grande quantidade de combinações derivadas da hierarquia e dos operadores temporais;
- A simulação pode ser utilizada como uma documentação interativa para explicar ou mostrar aos usuários finais como o sistema funciona.

2.4. Geração de Interfaces do Usuário

É possível criar um sistema interativo de maneira que sua interface tenha uma correspondência direta com o modelo da tarefa. E o suporte computacional para esse tipo de atividade também é possível. Nesse caso, é importante considerar não só as tarefas, mas também os objetos que podem ser manipulados durante a execução das mesmas.

3. Como Estão as Ferramentas Atualmente?

A maioria das ferramentas de suporte à análise e modelagem da tarefa da atualidade ainda são rudimentares. Ou não foram terminadas ou ainda possuem falhas que as impossibilitam de serem utilizadas por grupos que não sejam aqueles que as desenvolveram.

Funcionalmente, elas também ainda são imaturas. Muitas oferecem características que as limitam para a atividade de análise. Algumas das características que muitas não possuem é a capacidade de modelagem de sistemas cooperativos e multi-usuários.

Por essas razões, muitas das ferramentas existentes não foram liberadas para serem utilizadas por grupos externos. Ou se foram, ainda não se encontram num nível de maturidade

suficiente para terem sucesso. No entanto, algumas ferramentas mais recentes se destacam e se encontram num nível de utilização bastante satisfatório, como as que vamos discutir a seguir.

4. CTTE

Task Lotos Interactors Modeling (TLIM) (Paternò 1999) é um método de modelagem que inclui ConcurTaskTrees (Paternò et al. 1997), para a construção de modelos. ConcurTaskTrees são modelos de tarefas hierárquicos melhorados com operadores baseados em LOTOS para a especificação de restrições temporais entre tarefas. A ferramenta CTTE é um editor gráfico implementado em Java para a manipulação de ConcurTaskTrees. As tarefas têm um certo tipo, que é visualizado utilizando-se diferentes ícones, e a notação LOTOS é utilizada para especificar as restrições de tempo. As tarefas e os objetos utilizados nas mesmas podem ser descritos. Além da capacidade de edição de modelos, a ferramenta oferece também verificação de consistência, opções de métricas, simulação e avaliação de performance. Essa ferramenta pode ser encontrada no *site* <http://giove.cnuce.cnr.it/ctte.html>.

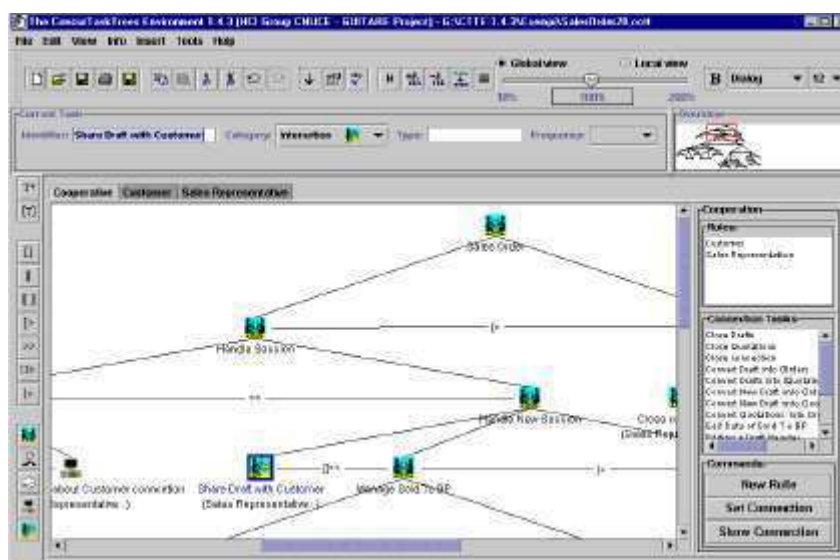


Figura 2: CTTE

5. Euterpe

Euterpe é um editor gráfico de modelos de tarefa desenvolvido em Prolog (funcionalidade) e C++ (GUI). Apesar de oferecer as funcionalidades principais de uma ferramenta de análise e modelagem da tarefa, que é o próprio suporte à modelagem da tarefa, ainda existem algumas funcionalidades a serem adicionadas a Euterpe, tais como: suporte à simulação e prototipação. As representações incluem árvores de tarefa, *templates*, e outras representações hierárquicas (Welie, Veer e Eliëns, 1998a, Welie, Veer, Eliëns, 1998b).

Algumas das funcionalidades que EUTERPE oferece são:

- Suporte à análise do modelo. Por exemplo, métricas;
- Suporte ao projeto em equipes;
- Suporte à documentação, ou seja, é possível gerar uma documentação impressa dos modelos construídos;
- Suporte à modelagem de diálogos.

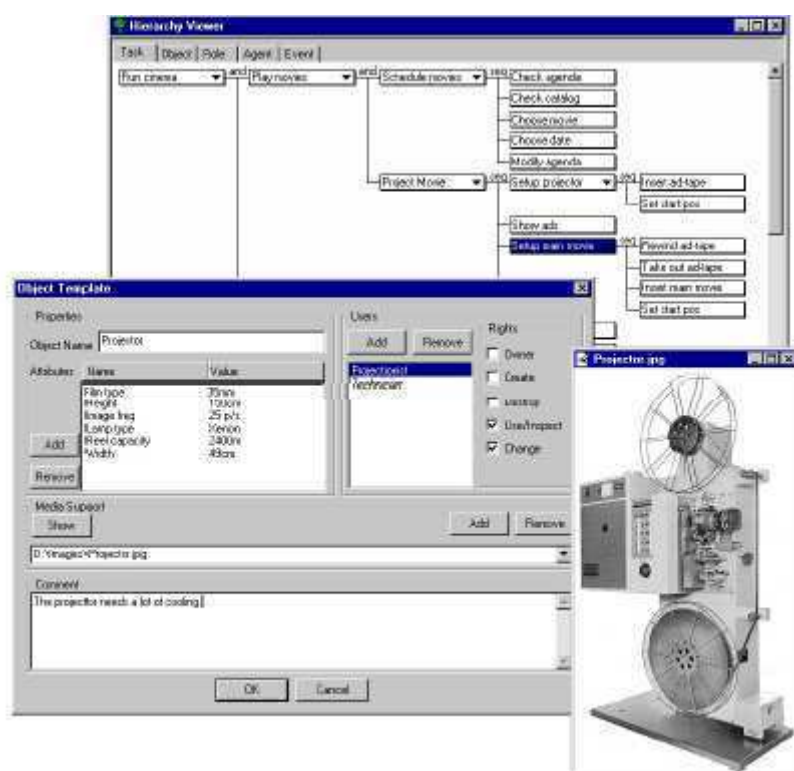


Figura 3: EUTERPE

6. IMAD*

IMAD* é uma ferramenta que dá suporte a MAD* e foi desenvolvida em C++. Ela permite a modelagem com o formalismo MAD* e oferece também alguns *templates* e construtores de tarefas. Trata-se de um editor gráfico de tarefas com suporte à verificação das descrições. A ferramenta identifica qualquer mudança na descrição e atualiza automaticamente todas as estruturas lógicas e visuais. Os modelos criados com o IMAD* podem ainda ser traduzidos em elementos de interface através de ISSI como um próximo passo rumo a um protótipo. A ferramenta está descrita completamente em francês o que dificulta a propagação de sua utilização (Gamboa, Scapin, 1997).

7. Funcionalidades de iTAOS

A partir das funcionalidades comentadas anteriormente e das funcionalidades que cada uma das ferramentas acima oferece, deduzimos as funcionalidades da ferramenta iTAOS, que são implementadas pelo módulo TAME.

Em sua concepção, a ferramenta iTAOS permite primeiro a modelagem da tarefa e algumas funcionalidades tais como: verificação da coerência e completude das informações, geração de documentação e reuso de modelos. Todas essas funcionalidades são detalhadas no Capítulo 6.

A Tabela 1 abaixo apresenta uma matriz de comparação entre as ferramentas comentadas neste capítulo e a ferramenta iTAOS:

	CTTE	Euterpe	IMAD*	iTAOS
Tarefas Concorrentes	X	X	X	X
Avaliação de Desempenho	X			
Métricas	X	X		
Verificação de Coerência e Completude	X			X
Geração de Documentação		X		X
Simulação	X			
Multi-plataforma	X			X

Tabela 1: Matriz de Comparação das Ferramentas

8. Conclusão

Durante este capítulo apresentamos as principais características de uma ferramenta de suporte à análise e modelagem de tarefas. Mostramos o estado atual desse tipo de ferramenta e comentamos a respeito de seu nível de maturidade. Comentamos a respeito de algumas ferramentas em nível de funcionalidades oferecidas e disponibilidade. E, por fim, apresentamos as funcionalidades da ferramenta iTAOS com uma breve comparação entre o que ela oferece e o que as outras ferramentas oferecem por meio de uma matriz de comparação.

Capítulo 3 – A Linguagem TAOS

1. Introdução

A linguagem TAOS (Medeiros e Rousselot, 1995, Medeiros, 1998a, Medeiros, 1998b, Medeiros, Kafure e Lula, 2000) é definida pelo módulo TAME, que também define uma metodologia descendente-ascendente de construção do modelo do domínio. Tal linguagem possui uma sintaxe, uma semântica e uma terminologia que atentam para os aspectos estáticos e dinâmicos presentes no domínio. Neste capítulo, será feita uma apresentação do meta-modelo definido pela linguagem TAOS.

2. Meta-Modelo TAOS

A linguagem TAOS propõe um modelo de representação que considera tanto o comportamento estático quanto o comportamento dinâmico de um dado domínio através de dois tipos de entidades ou conceitos: os conceitos estáticos (objetos, métodos e situações) e os conceitos dinâmicos (processos, ações e planos). Os conceitos estáticos representam entidades que não mudam de estado durante um intervalo de tempo considerável, ao contrário dos conceitos dinâmicos, que podem sofrer mudanças em um determinado intervalo de tempo.

A Figura 2 apresenta o meta-modelo da linguagem TAOS, na notação UML (*Unified Modeling Language*), mostrando toda a hierarquia de conceitos estáticos e dinâmicos, na forma de classes e subclasses, bem como as informações que cada um dos conceitos é capaz de guardar consigo, seus atributos/descriptores.

No topo da hierarquia está a classe *Concept*, que representa os conceitos do meta-modelo em seu mais alto nível. Tal classe possui três atributos:

1. *name* – nome do conceito.
2. *description* – uma descrição para o conceito.
3. *additionalAttributes* – uma mapa que relaciona um atributo adicional a seu respectivo valor. A todo conceito do meta-modelo TAOS podem ser adicionados atributos adicionais de acordo com as necessidades que o domínio a ser modelado exija.

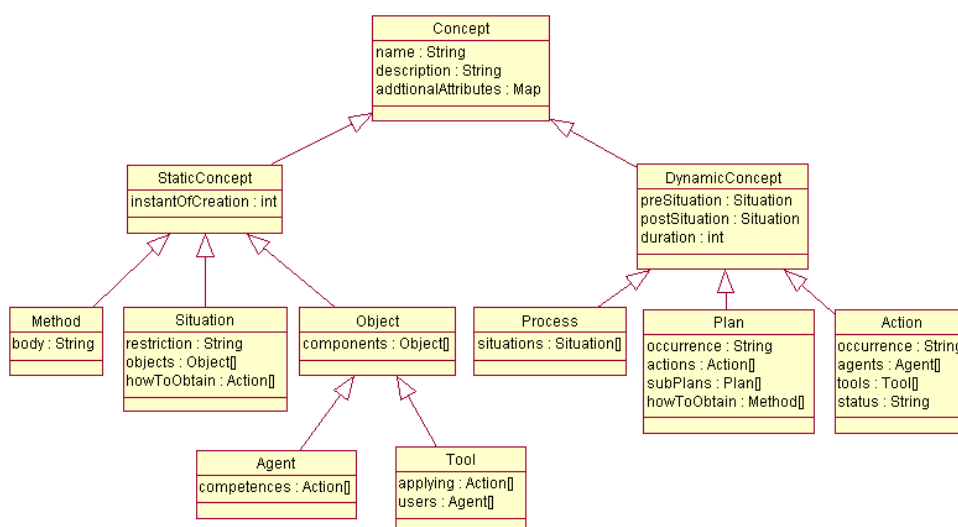


Figura 4: Meta-Modelo da Linguagem TAOS em UML

3. Conceitos Estáticos

Como subclasse de *Concept*, temos a classe *StaticConcept*, que representa os conceitos estáticos do meta-modelo. Além dos atributos herdados da classe *Concept*, essa classe tem ainda um atributo chamado *instantOfCreation*, que representa o instante de criação do conceito em uma possível tarefa de simulação (com suporte computacional) de um modelo de tarefas.

Como subclasses da classe *StaticConcept* podemos identificar mais três classes: *Object*, *Method* e *Situation*. Sendo a classe *Object* superclasse de mais duas classes: *Agent* e *Tool*.

A classe *Object* é utilizada para definir um objeto ou entidade envolvida na execução de uma ação, representando os objetos do domínio a ser modelado. Além dos atributos herdados, a classe *Object* tem ainda um atributo chamado *components*, que é um vetor de objetos que podem compor o objeto, ou seja, seus componentes.

A classe *Agent* define entidades capazes de executar ações do domínio modelado. O agente realiza ações que ele conhece e para as quais é competente e pode eventualmente tomar decisões a partir de informações que recebe de outros agentes. Além dos atributos herdados, a classe *Agent* possui ainda o atributo *competences*, que é um vetor de ações que o agente é capaz de realizar. Ações para as quais o agente é competente.

A classe *Tool* define entidades que são utilizadas pelos agentes para executar suas ações, ou seja, as ferramentas que podem ser empregadas para a execução de determinadas ações. Uma ferramenta deve ser manipulada apenas pelos agentes autorizados a utilizá-las. Além dos atributos herdados, a classe *Tool* possui ainda mais dois atributos:

1. *applying* – vetor de ações nas quais a ferramenta pode ser utilizada.
2. *users* – vetor de agentes habilitados a utilizar a ferramenta.

A classe *Method* define o método ou estratégia para executar um plano de tarefas. Além dos atributos herdados de suas superclasses, essa classe tem um atributo chamado *body*, que é o corpo do método. O corpo do método é uma expressão composta ou simples, que utiliza os operadores SEQ, OR, XOR, AND e SIM para descrever a decomposição do plano em sub-planos ou ações, segundo a seguinte gramática:

```

<expression> ::= <opr> (<list-expression>)
<list-expression> ::= <action>,<list-expression>
<list-expression> ::= <plan>,<list-expression>
<list-expression> ::= <expression>,<list-expression>
<list-expression> ::= <action>,<simple-expression>
<list-expression> ::= <plan>,<simple-expression>
<list-expression> ::= <expression>,<simple-expression>
<simple-expression> := <action>|<plan>|<expression>
<opr> ::= SEQ|OR|XOR|AND|SIM|PAR
<plan> ::= object of the type Plan
<action> ::= object of the type Action

```

Os elementos <plan> e <action> representam elementos das classes *Plan* e *Action*, que serão definidas mais adiante, na seção de conceitos dinâmicos.

Os operadores SEQ, OR, XOR, AND, SIM e PAR permitem estabelecer relações temporais (precedência e/ou concorrência) e/ou lógicas entre os sub-planos e/ou ações que compõem um plano. Abaixo temos a descrição de cada um dos operadores citados:

1. SEQ – indica que os sub-planos e/ou ações de um plano devem ser executados em seqüência.
2. OR – indica que pelo menos um dos sub-planos e/ou ações de um plano deve ser executado.
3. XOR – indica que apenas um dos sub-planos e/ou ações de um plano deve ser executado.

4. AND – indica que todos os sub-planos e/ou ações de um plano devem ser executados, não importando a ordem de execução dos mesmos.
5. SIM – indica que os sub-planos e/ou ações de um plano podem ser executados simultaneamente.
6. PAR – indica que os sub-planos e/ou as ações de um plano podem ser executados concorrentemente.

A classe *Situation* define o estado do mundo (objetos e as restrições sobre os objetos) em um determinado instante e as maneiras de chegar a esse estado. Além dos atributos herdados, essa classe contém os seguintes atributos:

1. *objects* – vetor de objetos sobre os quais a situação define as restrições.
2. *restriction* – expressão lógica que indica o estado no qual os objetos presentes na situação devem se encontrar, ao término ou início de uma tarefa, dependendo se a situação é uma pré-situação ou uma pós-situação.
3. *howToObtain* – vetor de planos ou ações que indicam as diversas maneiras como a situação pode ser atingida.

4. Conceitos Dinâmicos

Um conceito é considerado dinâmico se ele representa uma evolução de uma situação observada dentro de um intervalo de tempo. Essa evolução pode ser ocasionada pela intervenção intencional de um agente ou pela resposta (reação) automática de um artefato. A modelagem dessa evolução pode ser expressa a partir de um plano geral (mais alto nível de abstração) que pode ser decomposto em sub-planos e ações, até chegar a um nível de decomposição que contenha apenas ações. Esse tipo de conceito é representado pela classe *DynamicConcept*.

A classe *DynamicConcept* herda seus atributos da classe *Concept* e contém mais três atributos:

1. *preSituation* – define a situação inicial, necessária para o início da execução de um conceito dinâmico.
2. *postSituation* – define a situação resultante da execução de um conceito dinâmico.
3. *duration* – tempo de duração do conceito dinâmico.

A classe *Process* é um conceito dinâmico e define um conjunto de situações observadas em instantes diferentes (dispostas em ordem cronológica parcial ou total) e permite registrar a execução de um plano. Além dos atributos herdados, a classe *Process* tem o atributo *situations*, que é um vetor das situações que devem ser observadas.

A classe *Plan* define um plano de realização da tarefa e pode ser decomposto em sub-planos e/ou ações. Um plano pode ser decomposto em vários níveis de abstração, desde a sua nomeação (mais alto nível) até as ações elementares, e pode ser realizado de diversas maneiras (métodos). A classe *Plan* herda os atributos das classes *Concept* e *DynamicConcept* e ainda tem mais quatro atributos descritos abaixo:

1. *howToObtain* – vetor de métodos que impõem regras que indicam como os sub-planos/ações devem ser executados, impondo relações temporais e/ou lógicas entre eles.
2. *actions* – vetor de ações a serem executadas, juntamente com os sub-planos, para atingir o objetivo do plano, de acordo com as condições impostas pelos métodos que definem sua execução.
3. *subPlans* – vetor de sub-planos que devem ser executados, juntamente com as ações, para atingir o objetivo do plano, de acordo com as condições impostas pelos métodos que definem sua execução.
4. *occurrence* – define quantas vezes o plano deve ser executado, podendo assumir os seguintes valores:
 - a. (0,0) – o plano deve ser executado nenhuma vez.
 - b. (0,1) – o plano deve ser executado nenhuma vez ou uma vez.
 - c. (0,n) – o plano deve ser executado zero ou mais vezes.
 - d. (1,1) – o plano deve ser executado uma única vez.
 - e. (1,n) – o plano deve ser executado pelo menos uma vez.

A classe *Action* descreve uma tarefa elementar (ação). Utilizando a nomenclatura de árvores, na estrutura hierárquica de um plano, as ações são as folhas da árvore e não podem ser mais decompostas. A classe *Action* herda atributos da classe *Concept* e *DynamicConcept*, e tem ainda mais quatro atributos descritos abaixo:

1. *occurrence* – mesma definição encontrada na classe *Plan*.
2. *agents* – vetor de agentes competentes para realizar a ação.

3. *tools* – vetor de ferramentas que podem ser utilizadas pelos agentes para realizar a ação.
4. *status* – indica o estado da ação, podendo assumir os valores: em execução, finalizada, em espera, interrompida ou saltada.

5. Extensão de TAOS

Na seção anterior, a linguagem TAOS foi apresentada na forma em que foi concebida. Após ter sido validada como formalismo para descrição e análise de tarefas segundo MAD (Kafure, 2000), TAOS foi estendida para dar maior suporte à concepção de interfaces do usuário. A primeira coisa a ser feita foi mudar o nome da classe *Plan* para *Task*, que é a nomenclatura adotada pelos projetistas de interface. Agora a árvore construída com o formalismo não é mais composta de plano, sub-planos e ações, e sim de tarefa, sub-tarefas e ações. A segunda medida foi adicionar atributos orientados à concepção da interface do usuário, baseando-se em MAD* (extensão de MAD) e que serão apresentados mais adiante. De agora em diante, quando nos referirmos a TAOS, estaremos nos referindo a TAOS estendido. A priori, tais mudanças podem ser visualizadas na Figura 3, abaixo:

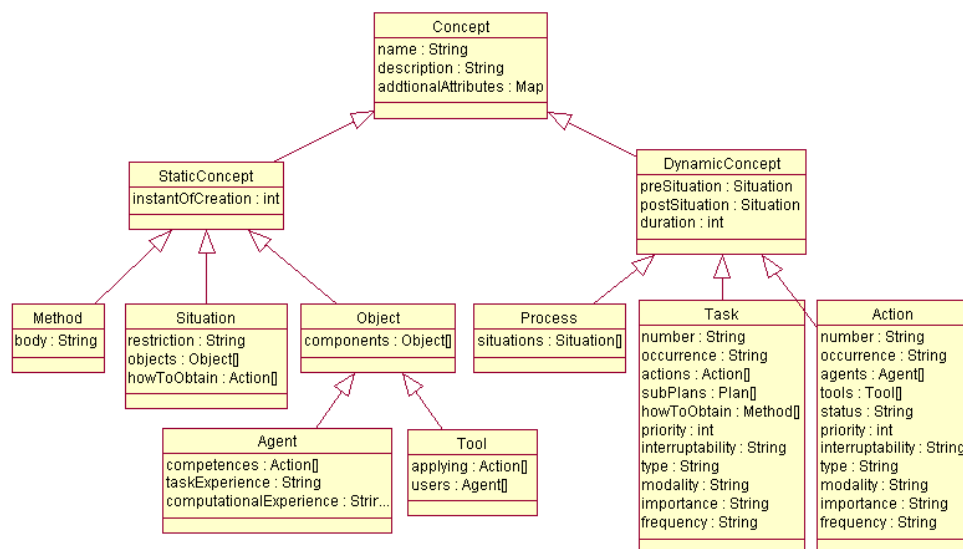


Figura 5: Meta Modelo da Linguagem TAOS Estendida

As classes *Task* e *Action* receberam os seguintes atributos:

1. *number* – número que identifica a tarefa na árvore de tarefas em que ela se encontra. Tal número é composto por um ou mais algarismos separados por

pontos (*i.e.* 1.1.2). A partir do número de uma tarefa é possível saber sua localização na árvore de tarefas. Uma tarefa de número 1.1.2 está localizada no terceiro nível da árvore e ocupa a segunda posição entre as sub-tarefas da tarefa de número 1.1.

2. *priority* – indica a prioridade da tarefa no modelo de tarefas. Esse atributo é um número inteiro. Sendo assim, uma tarefa com prioridade 1 tem preferência sobre uma tarefa de prioridade 0.
3. *interruptability* – interruptabilidade da tarefa, esse atributo pode assumir os seguintes valores:
 - a. Não Interrompível – para tarefas que não podem ser interrompidas.
 - b. Interrompível com reinício a partir do começo – se a tarefa for interrompida, o fluxo de sua execução deve ser retomado desde seu início.
 - c. Interrompível com reinício em curso – se a tarefa for interrompida, seu fluxo de execução pode ser retomado do ponto onde ela foi interrompida.
4. *type* – tipo da tarefa. Esse atributo assume valores de acordo com os meios de interação, tais como:
 - a. Sensório-Motora – para tarefas que são executadas a partir da interação do usuário com algum elemento, tal como um botão ou menu.
 - b. Mental – para tarefas tais como decidir e estimar.
 - c. Verbal – tarefas que podem ser ativadas verbalmente.
5. *modality* – modalidade da tarefa. Indica se a tarefa é automática, manual ou interativa.
6. *importance* – importância da tarefa, podendo assumir os valores: alta, média ou baixa.
7. *frequency* – indica o quão frequentemente a tarefa é executada, podendo assumir os valores: alta, média e baixa.

Outra classe que recebeu atributos orientados à concepção da interface do usuário foi a classe *Agent*. Os novos atributos de *Agent* são descritos abaixo:

1. *taskExperience* – experiência que o agente tem com a ação a ser executada por ele, podendo esta ser alta, média ou baixa.

2. *computationalExperience* – experiência que o agente tem com computadores ou com sistemas similares ao que está sendo modelado para dar suporte à sua tarefa. Essa experiência pode ser alta, média ou baixa.

6. Exemplo

Para um melhor entendimento do que é uma modelagem da tarefa do usuário realizada com a linguagem TAOS, mostraremos nesta seção um extrato do modelo de tarefas obtido durante o início do processo de desenvolvimento do módulo TAME. Tal modelo foi obtido através de entrevistas dirigidas com pessoas da área de interfaces homem-máquina da UFCG que utilizam a linguagem TAOS. Nosso exemplo consiste de um extrato da árvore de tarefas do modelo da tarefa: “Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas” (Medeiros, Cordeiro e Lula, 2002a) como ilustrado na Figura 4, abaixo:

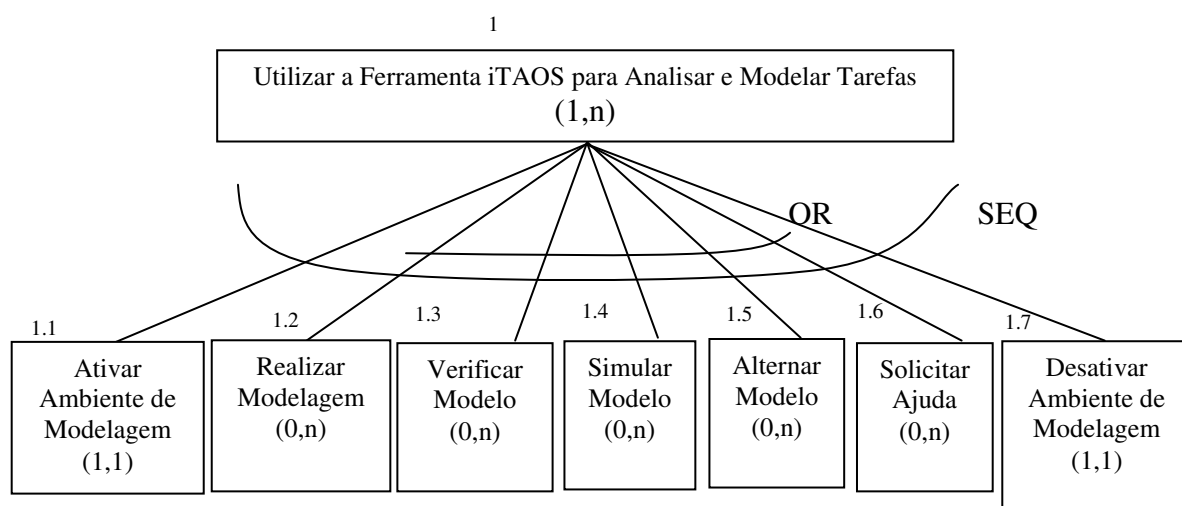


Figura 6: Extrato da tarefa “Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas”

Podemos observar os dois primeiros níveis da árvore de tarefas. No primeiro nível está a tarefa em questão e no segundo nível estão as ações: “Ativar Ambiente de Modelagem” e “Desativar Ambiente de Modelagem”; e as sub-tarefas: “Realizar Modelagem”, “Verificar Modelo”, “Simular Modelo”, “Alternar Modelo” e “Solicitar Ajuda”.

A seguir podemos ver os descritores de alguns conceitos do modelo em questão. O Descritor 1 refere-se à tarefa raiz da árvore de tarefas. A seguir, nos Descritores 2 e 3 podemos observar as informações a respeito de sua pré-situação e de seu método, respectivamente.

O Descritor 4 refere-se à ação “Ativar Ambiente de Modelagem”. Para finalizar, o Descritores 5 descreve um agente capacitado a realizar a ação e o Descritor 6 descreve um instrumento que pode ser utilizado para realizá-la.

De acordo com a notação da linguagem TAOS e obedecendo a sua semântica, podemos concluir a partir do modelo mostrado na Figura 4, que a tarefa em questão deve ser executada como uma seqüência de sub-tarefas e/ou ações (operador SEQ). Sendo que as sub-tarefas de 1.2 a 1.6 também estão sob a ação do operador OR, indicando que elas podem ou não ser executadas, mas se forem executadas, devem obedecer à seqüência.

Classe	Tarefa
Nome	Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas
Número	1
Descrição	Modelar tarefas utilizando a ferramenta iTAOS
Pré-situação	Situação Inicial de Modelar Tarefa
Pós-situação	Situação Objetivo de Modelar Tarefa
Ocorrência	(1,n)
Ações	[Ativar Ambiente de Modelagem, Desativar Ambiente de Modelagem]
Sub-Tarefas	[Realizar Modelagem, Verificar Modelo, Simular Modelo, Alternar Modelo, Solicitar Ajuda]
Como Realizar	Método de Modelar Tarefas
Prioridade	() 1 () 2 (x) 3
Interruptibilidade	() Não interrompível () Interruptível com reprise no início (x) Interruptível com reprise em curso
<i>ATRIBUTOS ORIENTADOS A ESPECIFICAÇÃO DA IHM</i>	
Tipo da Tarefa	(x) sensório-motora (x) mental (x) verbal Outro:
Modalidade da tarefa	() manual () automática (x) interativa
Importância da Tarefa	(x) alta () média () baixa
Frequência da Tarefa	(x) alta () média () baixa

Descritor 1: Descritores da tarefa raiz

Classe	Situação
Nome	Situação Inicial da Tarefa Raiz
Descrição	Situação que deve ser satisfeita para a realização de uma modelagem da tarefa com a ferramenta iTAOS.
Objetos	[Ambiente de Modelagem, Projetista de Interface, Mouse, Teclado]

Restrição	Disponível(Ambiente de Modelagem) AND Disponível(Projetista de Interface) AND Disponível(Mouse) AND Disponível(Teclado)
Como Atingir	

Descritor 2: Descritores da pré-situação da tarefa raiz

Classe	Método
Nome	Método da Tarefa Raiz
Descrição	Estratégia para modelar tarefa com a ferramenta iTAOS
Corpo	SEQ(Ativar Ambiente de Modelagem, OR(Realizar Modelagem, Verificar Modelo, Simular Modelo, Alternar Modelo, Solicitar Ajuda, Desativar Ambiente de Modelagem))

Descritor 3: Descritores do método da tarefa raiz

Classe	Ação
Nome	Ativar Ambiente de Modelagem
Número	1.1
Descrição	Inicializa o Ambiente de modelagem
Pré-situação	Situação Inicial de Modelar Tarefa
Pós-situação	Situação Objetivo de Ativar Ambiente de Modelagem
Ocorrência	(1,1)
Agente	Projetista de Interface
Instrumento	Ambiente de Modelagem, Mouse
Prioridade	() 1 () 2 (x) 3
Interruptibilidade	(x) Não interrompível () Interruptível com reprise no início () Interruptível com reprise em curso
ATRIBUTOS ORIENTADOS A ESPECIFICAÇÃO DA IHM	
Tipo da tarefa	(x) sensório-motora () mental () verbal Outro:
Modalidade da Tarefa	(x) manual () automática () interativa
Importância da Tarefa	(x) alta () média () baixa
Frequência da Tarefa	() alta () média (x) baixa

Descritor 4: Descritores da ação: Ativar Ambiente de Modelagem

Classe	Agente
Nome	Projetista de Interface
Descrição	Projetista de interface que modela tarefas
Competência	Todas as ações da tarefa raiz
Experiência com a Tarefa	() alta (x) média () baixa
Experiência Computacional	(x) alta () média () baixa

Meios de Interação	
---------------------------	--

Descritor 5: Descritores do agente: Projetista de Interface

Classe	Ferramenta
Nome	Mouse
Descrição	Ferramenta utilizada para ativar o ambiente de modelagem
Ator	Projetista de Interface
Emprego	Ativar Ambiente de Modelagem e outras ações

Descritor 6: Descritores da ferramenta: Mouse

7. Conclusão

Neste capítulo, foi feita a apresentação dos conceitos da linguagem TAOS e dos atributos que compõem cada um deles. Mostramos também que a linguagem TAOS foi estendida para dar maior suporte à análise e modelagem da tarefa, já que TAOS foi validado como formalismo para análise e modelagem de tarefas segundo MAD. Tal extensão compreende a inclusão de atributos orientados à concepção de interfaces a alguns de seus conceitos e a mudança de nome da classe *Plan* para *Task*. E é justamente a linguagem TAOS estendida que será implementada em vista da construção da ferramenta iTAOS.

Para mostrar como os conceitos são aplicados foi mostrado um exemplo com figuras que representam um extrato de um modelo de tarefas, consistindo de uma parte da árvore de tarefas e dos descritores de alguns conceitos presentes no modelo.

Capítulo 4 – O Processo de Desenvolvimento

1. Introdução

Este capítulo descreve todo o processo de desenvolvimento do módulo TAME, contendo informações a respeito do processo de desenvolvimento de *software* escolhido, bem como as razões da escolha do mesmo. O capítulo contém também informações a respeito do fluxo de trabalho (*workflow*) seguido em cada fase do processo e das atividades mais relevantes que foram executadas em cada um desses fluxos, bem como dos artefatos obtidos.

2. Escolha do Processo

A escolha do processo de desenvolvimento do módulo TAME foi baseada em nossas hipóteses de trabalho. O módulo TAME foi desenvolvido em um processo separado do processo de desenvolvimento do módulo TAOS-Graph, segundo o princípio da independência do diálogo, primeira hipótese.

Baseado em nossa segunda hipótese, que diz que é possível deduzir os requisitos funcionais de um sistema computacional a partir da análise e modelagem da tarefa do usuário para o qual o sistema se destina, o processo de desenvolvimento do módulo TAME, que implementa as funcionalidades da ferramenta iTAOS, foi baseado, justamente, na análise e modelagem da tarefa.

Apesar de independentes, não faria sentido iniciar o processo de desenvolvimento dos dois módulos a partir de análises de requisitos separadas, ou seja, com requisitos levantados a partir de atividades de análise independentes, pois isso poderia levar a uma incoerência entre os requisitos levantados para cada um dos módulos. Em outras palavras, algumas funcionalidades previstas para o módulo TAME poderiam não coincidir com algumas funcionalidades previstas para o módulo TAOS-Graph e vice versa, impossibilitando ou tornando mais difícil uma futura integração entre os dois módulos para formar a ferramenta iTAOS. Esse talvez seja um ponto fraco do princípio da independência do diálogo. Se

quisermos desenvolver a parte funcional de um sistema em um processo paralelo ao processo de desenvolvimento da interface do usuário, é importante que os dois processos possuam um ponto de partida em comum e alguma maneira de permitir a comunicação entre os processos, de modo a permitir ajustes de um ou de outro de acordo com a necessidade, e sua integração ao final.

Sendo assim, foi decidido que os processos partiriam da análise e modelagem da tarefa: “Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas”. E para a comunicação e a integração, foi decidido que, no processo do módulo TAME, a primeira coisa a ser feita após a análise da tarefa seria a identificação dos objetos do domínio, o projeto das classes, e a definição dos serviços que tais classes disponibilizariam para o módulo TAOS-Graph através de uma API (Application Program Interface). É justamente essa API, o ponto de comunicação e integração entre os dois módulos. A API define uma interface geral de interação com o módulo TAME.

Nossa terceira hipótese considera que o Processo Unificado poderia dar suporte ao processo de desenvolvimento do módulo TAME. Porém, ao tentar aplicá-lo ao nosso caso vimos que não seria possível utilizá-lo rigorosamente, pois ele é completamente baseado em *use cases* e não faria sentido fazer também *use cases* se já tínhamos tarefas, pois ambos fornecem as mesmas informações. No entanto, utilizamos alguns elementos do Processo Unificado, tais como, fases, iterações e fluxo de atividades, de uma maneira adaptada, para seguir o processo de desenvolvimento do módulo TAME. A seguir, veremos algumas informações importantes a respeito do Processo Unificado, que serviu de base para o nosso processo de desenvolvimento.

3. O Processo Unificado

Esta seção fornece uma visão do que é o Processo Unificado sem entrar em detalhes que fujam do escopo deste trabalho. O Processo Unificado é um processo de *software*, pois define quem faz o que, quando fazer, como atingir um certo objetivo e as entradas e saídas de cada atividade para o desenvolvimento de *software*. E é também um *framework*, pois provê as entradas e saídas de cada atividade, mas não restringe como cada atividade deve ser realizada.

O processo Unificado engloba atividades de baixo nível (tal como achar classes), que são combinadas em fluxos de atividades, que descrevem como cada atividade fornece informações para outra. Os fluxos de atividades são organizados em iterações e cada iteração

identifica alguns aspectos do sistema a serem considerados. As iterações são organizadas em fases, que têm foco em um aspecto do processo. Para finalizar, fases podem ser agrupadas em ciclos. Cada ciclo tem foco na geração de versões sucessivas de um sistema (versão 1.0, versão 1.1, etc.). Tudo isso pode ser visualizado na Figura 5, abaixo:

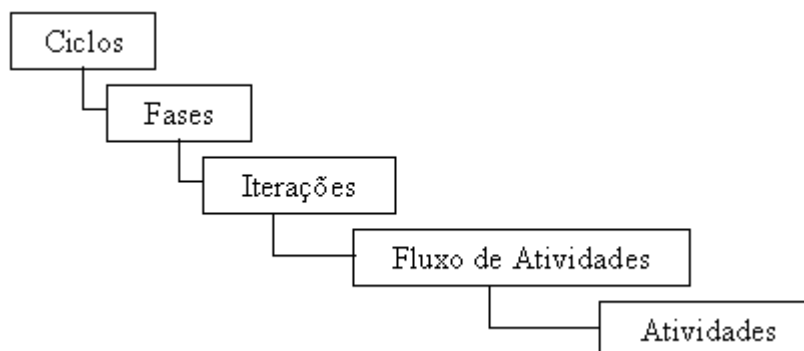


Figura 7: Blocos chave do Processo Unificado

3.1. Características do Processo Unificado

O Processo Unificado possui quatro elementos chave:

- É um processo iterativo e incremental.
- É baseado em *use cases*.
- É centrado na arquitetura.
- Considera a existência de riscos de projeto.

Quando dizemos que o Processo Unificado é iterativo e incremental, estamos dizendo que ele é baseado em iterações, onde em cada uma delas é tratado um aspecto diferente do processo. Alguns aspectos podem ser deixados para serem tratados mais à frente durante o processo. Essa característica de poder deixar alguns aspectos do sistema para serem tratados futuramente é a característica incremental do processo.

O Processo Unificado é também baseado em *use cases*. *Use cases* ajudam desde cedo a identificar quem usa o sistema e o que deve ser implementado pelo sistema (funcionalidades). No Processo Unificado, *use cases* são utilizados para assegurar que a evolução do projeto do sistema esteja sempre de acordo com o que o usuário requisitou. Deve existir a possibilidade de rastreamento entre o que foi implementado e os requisitos do sistema, e isso é atingido pelo uso de *use cases*.

O Processo Unificado reconhece explicitamente a necessidade de uma arquitetura, por ser centrado na arquitetura. Ele descreve como identificar, o que deve fazer parte de uma arquitetura e como deve ser feito o projeto e a implementação da mesma.

E quanto ao reconhecimento da existência de riscos, o Processo Unificado põe em destaque aspectos desconhecidos do sistema e outras áreas mais preocupantes, tratando-os como aspectos críticos que devem fazer parte da arquitetura ou como áreas de risco que devem ser tratadas no início do processo, momento em que se tem mais tempo, ao invés de deixar para o fim, quando geralmente o tempo é mais curto.

3.2. Ciclos de Vida

Um ciclo de desenvolvimento é o período de tempo medido desde o início do projeto até a entrega (ou cancelamento) de uma versão do mesmo. Um ciclo inclui todas as atividades executadas durante esse tempo.

Nem todos os ciclos de um projeto têm as mesmas características, à medida que o produto evolui, os ciclos de vida de cada versão tem foco em um objetivo diferente do anterior. Geralmente, durante o tempo de vida de um *software*, podemos distinguir ciclos de desenvolvimento inicial de ciclos evolutivos e ciclos de manutenção (Kruchten, 2002).

3.3. Fases de um Ciclo de Vida

Cada ciclo é dividido em uma seqüência de quatro fases: Planejamento, Elaboração, Construção e Transição, como ilustra a Figura 6, abaixo.

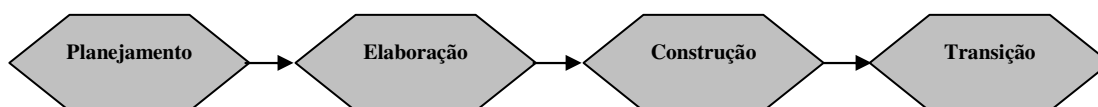


Figura 8: As quatro fases do Processo Unificado

Essas fases têm uma grande importância para o processo, não pelo que é executado, e sim pelo que é atingido como macro *milestone* ao final de cada uma delas.

Durante a fase de Planejamento é definido o escopo do projeto e é desenvolvido o *business case* para o sistema. É nessa fase que é discutida a viabilidade do projeto.

Durante a fase de Elaboração são capturados os requisitos funcionais do sistema. Qualquer requisito não-funcional pode ser identificado aqui. Uma outra tarefa que deve ser

executada durante essa fase é a criação da arquitetura a ser seguida durante o processo de *software*.

A fase de Construção se concentra em completar a análise do sistema, refinar o projeto e implementar o sistema. É durante essa fase que o produto é praticamente construído.

A fase de Transição põe o sistema à disposição do usuário. Essa fase envolve atividades tais como: entrega do sistema e manutenção do mesmo.

Ao final de cada fase é gerado um macro *milestone*. Cada um desses *milestones* engloba um conjunto de artefatos que constituem o produto de cada fluxo de atividades da fase. Os principais *milestones* para cada fase são:

- Planejamento – o resultado obtido durante essa fase é uma visão do sistema, que inclui um modelo de *use case* simplificado (para identificar as principais funcionalidades do sistema) e uma arquitetura candidata. É durante essa fase que os principais riscos são identificados e que a fase de elaboração é planejada.
- Elaboração – o principal artefato dessa fase é a arquitetura junto com um modelo de *use cases* detalhado e um conjunto de planos para a fase de construção.
- Construção – o resultado dessa fase é o produto implementado, o qual inclui o *software* bem como o projeto e os modelos associados. Nessa fase o produto pode ter alguns defeitos e algumas atividades a serem executadas durante a fase de transição.
- Transição – o principal artefato dessa fase é o produto final.

3.4. Iterações

Cada fase abrange uma ou mais iterações. O *software* é produzido em cada iteração, e cada uma delas é finalizada com um micro *milestone* incluindo uma versão (interna ou externa) que é um ponto de avaliação do progresso do projeto. O produto cresce de forma incremental à medida que as iterações ocorrem.

3.5. Fluxos de Atividades (Workflows)

O Processo Unificado possui cinco atividades básicas: Requisitos, Análise, Projeto, Implementação e Teste. As quatro fases do processo abrangem essas cinco atividades, porém,

cada um deles ocorre com uma intensidade diferente dependendo da fase em que o produto se encontra no processo. Por exemplo, durante a fase de Planejamento existe implementação (geralmente para construir protótipos), mas essa atividade é bem menos intensa nessa fase do que na fase de Construção. Sendo assim, podemos dizer que o Processo Unificado é um espiral, como mostrado na Figura 7, abaixo:

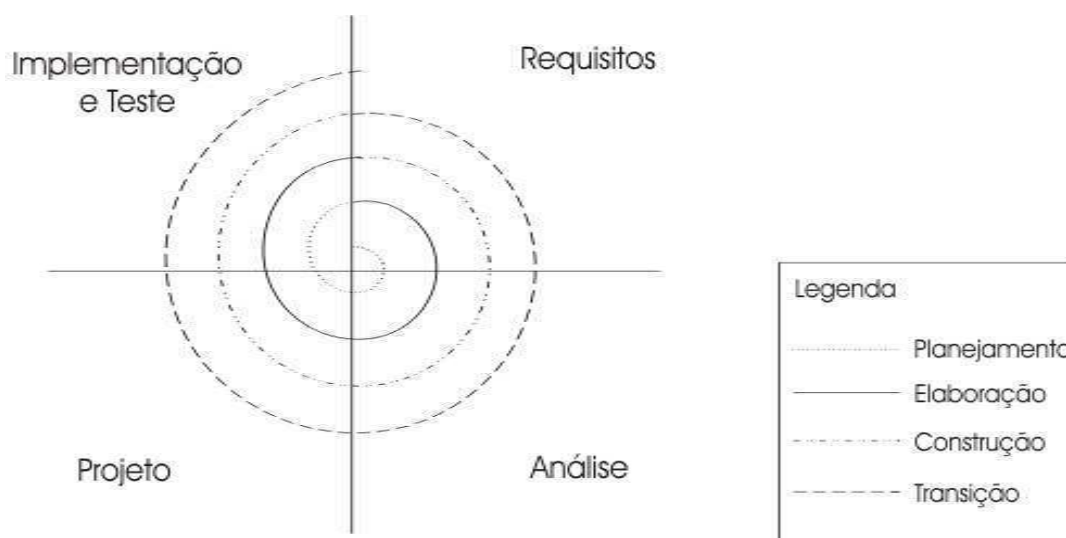


Figura 9: Fases e *workflows* do Processo Unificado

O levantamento de requisitos tem foco nas atividades de identificação dos requisitos funcionais e não-funcionais do sistema. O produto final dessa atividade é o modelo de *use cases*. Os requisitos identificados servem de entrada para a atividade de análise, onde são reestruturados em termos mais técnicos do *software* a ser construído, o que é obtido com tal reestruturação é um modelo de análise. O modelo de análise serve de entrada para o projeto, que transforma tal modelo em um modelo de projeto a ser implementado durante a implementação, que representa a codificação do projeto em uma linguagem apropriada, sua compilação, distribuição e geração da documentação. Em seguida, o *software* gerado durante o a implementação é testado para assegurar que todos os requisitos funcionais foram implementados, que o *software* é confiável, etc.

3.6. Atividades

Cada *workflow* abrange um conjunto de atividades principais que devem ser executadas durante cada um deles. As atividades principais de cada *workflow* podem ser vistas na tabela abaixo:

Atividade	Fluxo
Requisitos	<ul style="list-style-type: none"> • Identificar e descrever os atores do sistema. • Identificar os <i>use cases</i>. • Descrever os <i>use cases</i> apropriadamente. • Descrever o modelo de <i>use case</i>. • Preparar um glossário com os termos utilizados no sistema.
Análise	<ul style="list-style-type: none"> • Fazer uma análise arquitetural. • Identificar as classes do modelo de análise. • Organizar as classes identificadas em pacotes. • Fazer uma análise de <i>use cases</i> para gerar <i>use case realizations</i>
Projeto	<ul style="list-style-type: none"> • Fazer um projeto arquitetural. • Gerar classes de projeto. • Identificar as interfaces de projeto. • Gerar <i>use case realizations</i> de projeto a partir dos <i>use case realization</i> gerados no <i>workflow</i> de Análise. • Identificar subsistemas.
Implementação	<ul style="list-style-type: none"> • Implementar o esqueleto da arquitetura. • Definir o modelo de implementação. • Implementar subsistemas, classes e interfaces. • Integrar sistemas.
Teste	<ul style="list-style-type: none"> • Planejar os testes a serem feitos para sistema. • Projetar e implementar os casos de teste, criando casos de teste executável. • Executar os testes e analisar os resultados obtidos.

Tabela 2: Atividades do PU e seus principais fluxos de atividades

4. A Adaptação

A adaptação foi feita de maneira *ad hoc*. Nós aproveitamos a estrutura de fases e atividades que compõem o Processo Unificado. As mudanças ocorreram no fluxo de cada

atividade. A tabela abaixo, que é semelhante à Tabela 1, resume o fluxo de atividades seguido para cada atividade.

Atividade	Fluxo
Requisitos	<ul style="list-style-type: none"> • Identificar as tarefas. • Descrever as tarefas apropriadamente. • Descrever o modelo de tarefas. • Preparar um glossário com os termos utilizados no sistema.
Análise	<ul style="list-style-type: none"> • Fazer uma análise arquitetural. • Identificar as classes que devem fazer parte do sistema. • Organizar as classes identificadas em pacotes.
Projeto	<ul style="list-style-type: none"> • Fazer um projeto arquitetural. • Gerar classes de projeto. • Identificar as interfaces de projeto (API). • Identificar subsistemas.
Implementação	<ul style="list-style-type: none"> • Implementar o esqueleto da arquitetura. • Definir o modelo de implementação. • Implementar subsistemas, classes e interfaces. • Integrar sistemas.
Teste	<ul style="list-style-type: none"> • Planejar os testes a serem feitos para sistema. • Projetar e implementar os casos de teste, criando casos de teste executáveis. • Executar os testes e analisar os resultados obtidos.

Tabela 3: Fluxo de Atividades do Processo do Módulo TAME

5. Tarefas vs. *Use Cases*

Por que não utilizamos *use cases* ao invés de tarefas e seguimos o Processo Unificado como ele realmente é?

Em primeiro lugar, nosso trabalho é baseado em uma hipótese que diz que é possível levantar os requisitos de um sistema a partir da análise de tarefas, e se utilizarmos essa técnica seria redundante utilizar os *use cases*. Em segundo lugar, após uma comparação entre as duas

técnicas, apresentada a seguir, vimos que não existem razões que impeçam a utilização de uma ou de outra. Antes de mostrar as vantagens e desvantagens das tarefas e *use cases* vamos fazer uma breve definição do que são *use cases*.

5.1. Use Cases

Um *use case* é uma seqüência especial de transações realizadas por um usuário e um sistema por meio de um diálogo (Jacobson, 2003). *Use cases* compõem modelos de *use case* que servem para descrever sistemas de em uma perspectiva Black Box.

Se nos aprofundarmos em um modelo de *use cases* estamos, na realidade, entrando na descrição de seus *use cases*. Cada descrição contém as seguintes informações:

- Uma breve descrição dos propósitos do *use case*;
- Um fluxo de controle;
- Fluxos básicos e alternativos;
- Sub fluxos;
- Pré-condições e pós-condições.

Existem basicamente quatro tipos de *use cases*:

- Concretos – *use cases* que podem ser instanciados;
- Generalizações - *use cases* que suportam reuso de *use cases*;
- Extensões – *use cases* que adicionam comportamento a um *use case* existente sem mudar o *use case* original;
- Inclusões – *use cases* que adicionam comportamento a um *use case* alterando o *use case* original.

5.2. Semelhanças

A partir da definição de *use case* dada acima, percebemos que existe uma certa semelhança entre eles e tarefas. Assim já podemos começar a tirar conclusões acerca de modelos de tarefas a respeito da possibilidade de desenvolver sistemas a partir deles:

- Já que modelos de *use cases* são utilizados, na Engenharia de Software, para capturar requisitos de sistema, por que não podemos utilizar modelos de tarefas, já que ambos os modelos possuem uma certa semelhança.
- Ambos identificam os usuários do sistema, que são os atores no caso do modelo de *use cases* e o usuário no caso do modelo de tarefas.

- A partir dos dois modelos podemos obter informações a respeito do fluxo a ser seguido para que uma certa atividade seja executada (comportamento dinâmico do sistema). O modelo de tarefas fornece tais informações através de um único diagrama, enquanto que o modelo de *use cases* fornece tais informações de maneira distribuídas entre vários diagramas.

Um estudo a respeito das semelhanças e diferenças entre *use cases* e tarefas, bem como adaptações feitas a processos de desenvolvimento de *software* para dar suporte à análise e modelagem da tarefa em vista da concepção da interface do usuário, mostrando as vantagens e desvantagens de utilizar modelos de tarefas no lugar de modelos de *use cases*, podem servir de ponto de partida para trabalhos futuros.

5.3. Vantagens e Desvantagens

Podemos citar algumas vantagens e desvantagens em relação ao uso de tarefas ou *use cases* para desenvolver sistemas. Talvez a capacidade de representação de modelos de tarefas e modelos de *use cases* seja a mesma, contudo, a maneira de representação pode diferir em alguns aspectos.

Uma grande vantagem do uso de modelos de *use cases* é que eles oferecem uma linguagem com uma notação fácil de ser entendida pelo usuário ou cliente. Coisa que modelos de tarefas não oferecem.

Outra vantagem do modelo de *use cases* é que fica explícito no diagrama quem é o usuário, enquanto no modelo de tarefas essa informação só é encontrada quando lemos os descritores das tarefas. Por outro lado um modelo de tarefas contém bem mais informações a respeito do usuário, possibilitando que o sistema seja bem mais adequado ao tipo de usuário para o qual ele se destina.

Uma vantagem do modelo de tarefas é que é possível representar as tarefas do usuário e os fluxos das mesmas através de um único artefato, que é a árvore de tarefas do modelo de tarefas. Esse tipo de informação não fica a vista em um diagrama de *use cases*, é necessário complementar a descrição do *use case* com diagramas de seqüências.

Use cases propõem também um mecanismo de reuso de *use cases* por meio de relacionamentos, que podem ser vistos graficamente. Esse tipo de mecanismo é muito importante na atividade de análise, principalmente para economizar tempo. Alguns formalismos para modelagem de interface, entre eles o TAOS, também fornecem um

mecanismo semelhante. No caso do formalismo TAOS esse mecanismo é o atributo “Super-Conceito” presente em cada um dos conceitos definidos pela linguagem. Talvez esse mecanismo não seja tão forte como o dos *use cases*, mas também funciona.

Como vimos, não existe nenhuma disparidade tão grande a ponto de impedir que usássemos tarefas ao invés de *use cases*, ou vice-versa, e, portanto, prosseguimos nosso processo com o uso de tarefas, de acordo com nossa segunda hipótese.

6. Desenvolvimento do Módulo TAME

Esta seção descreve as fases do desenvolvimento do módulo TAME. A seguir são feitos alguns comentários a respeito de cada uma destas fases dando ênfase aos *macro milestones* obtidos ao final de cada uma delas e como eles foram atingidos.

6.1. Fase de Planejamento

Durante a fase de Planejamento realizamos as atividades de Requisitos e Análise iterativamente. Durante a atividade de Requisitos tentamos expressar os requisitos do sistema em termos de tarefas, e construímos o modelo de tarefas para a ferramenta iTAOS. Tal modelo foi construído através da atividade análise e modelagem da tarefa, realizada através de entrevistas semidirigidas com futuros usuários do sistema: projetistas de interfaces do grupo de Interfaces Homem Computador da UFCG e alunos de disciplinas que utilizam a linguagem TAOS para modelar tarefas. Esse fluxo de atividades foi executado de forma iterativa e incremental, sendo as entrevistas realizadas semanalmente, com o intuito de identificar novas funcionalidades a cada iteração até que chegamos ao consenso de que todas as funcionalidades tinham sido exploradas.

Durante a atividade de Análise utilizamos o modelo de tarefas para realizar a identificação das entidades principais do sistema. O próprio modelo de tarefas fornece tais entidades como objetos presentes nas pré e pós-situações das tarefas. Tais entidades foram então representadas como classes e organizadas em pacotes de acordo com suas características. Entre as entidades principais estão os conceitos do meta modelo TAOS e classes de manipulação dos conceitos, tais como: *Model*, para representar modelos de tarefa, e *TAOSTaskTree*, para representar a árvore de tarefas do modelo.

As entidades identificadas durante a Análise constituem a arquitetura base do módulo TAME, que fornece uma visão do que se trata o sistema (módulo TAME). Os diagramas de

classes das Figuras 8 e 9 mostram os relacionamentos entre as classes pertencentes à arquitetura base e o diagrama de pacotes da Figura 10 mostra como essas classes estão organizadas. Cada classe presente no diagrama tem um identificador que indica a que pacote ela pertence. Por exemplo, a classe *Task* possui um identificador “*from dynamicconcepts*”, informando que essa classe pertence ao pacote *dynamicconcepts*.

Essa arquitetura sofreu modificações evolutivas durante as fases seguintes em vista de aumentar sua capacidade de reuso e aumentar sua flexibilidade com relação a alterações que possam surgir futuramente no sistema.

O *macro milestone* dessa fase foi a arquitetura base do módulo TAOS, obtida a partir da análise do modelo de tarefas. Como artefatos adicionais foram gerados um documento de Visão, um Plano de Gerenciamento de Requisitos e um Glossário, que podem ser vistos nos Anexos A, B e C, respectivamente. Esse conjunto de artefatos tem o objetivo de nos dar uma visão geral do que se trata o sistema e de como seu desenvolvimento é acompanhado.

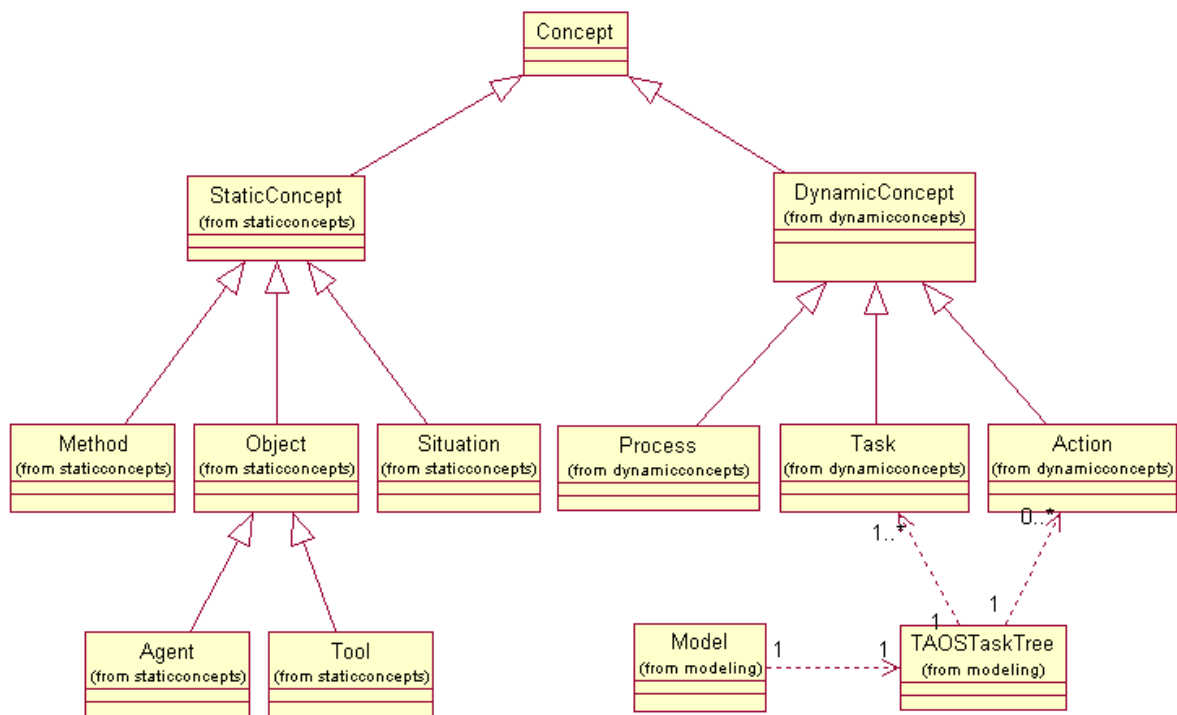


Figura 10: Arquitetura base do módulo TAME na notação UML

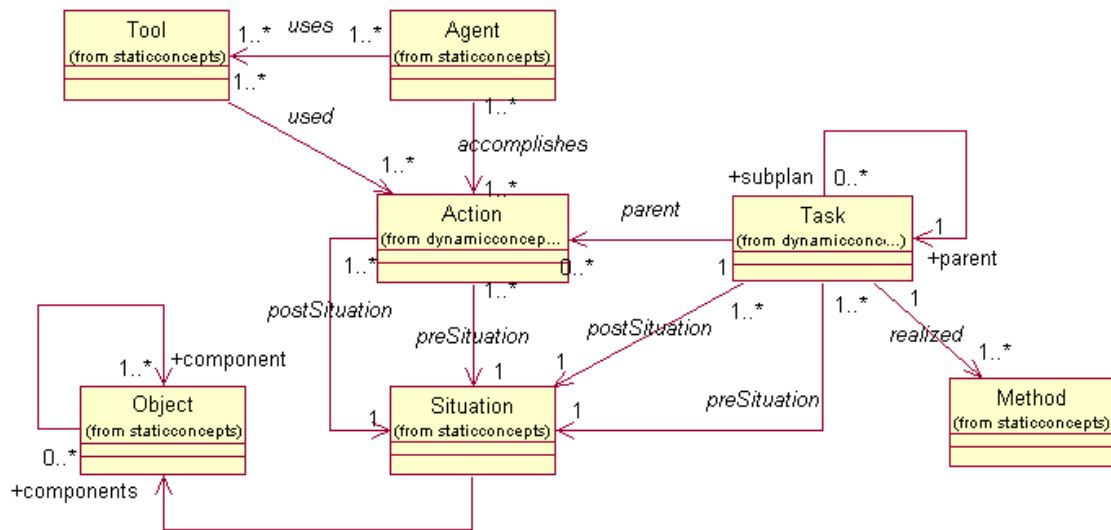


Figura 11: Relacionamento inter-classes da arquitetura base do módulo TAME

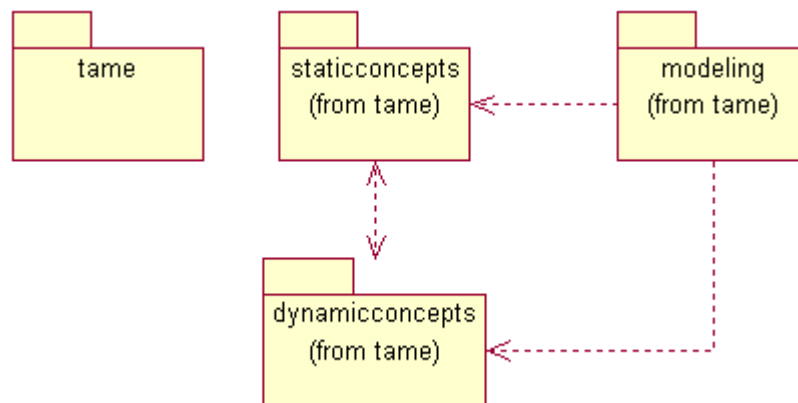


Figura 12: Estrutura de pacotes do módulo TAME

5.2. Fase de Elaboração

A atividade predominante nesta fase é a de Projeto. O projeto é dirigido à implementação e durante o mesmo foram identificadas as classes que compõem o projeto arquitetural do módulo TAME, suas responsabilidades, operações e relacionamentos.

O projeto arquitetural é uma junção da evolução da arquitetura base e algumas informações a respeito das decisões tomadas, tais como: linguagem de programação escolhida (Java), meio de persistência a ser utilizado e a escolha de padrões de projeto para aumentar o reuso das classes do projeto (Vlissides, Helm, Gamma *et al.*, 1994). E para o nosso caso, o

modelo de tarefas também entra na arquitetura, já que ele contém informações a respeito do comportamento dinâmico do sistema.

A persistência dos modelos de tarefas construídos com o uso do módulo TAME será feita em arquivos XML (Extensible Markup Language) (Goldfarb e Prescod, 2000), que é uma linguagem que vem sendo largamente utilizada para a representação de dados. Essa linguagem vem sendo cada vez mais usada mundialmente e é um padrão da W3C (World Wide Web Consortium). Outro motivo para a escolha desse meio de persistência é que a plataforma Java fornece facilidades para a manipulação de documentos XML.

Com a escolha de XML como meio de persistência dos modelos de tarefas, foi criado um DTD (Document Type Definition), que é um documento que contém informações a respeito da estrutura de elementos que os arquivos devem conter. É o DTD que contém informações capazes de indicar se o documento XML é válido ou não. Todo documento XML que represente um modelo de tarefas construído com o uso do módulo TAME deve satisfazer esse DTD.

Em vista de aumentar o reuso de código e facilitar futuras alterações, a arquitetura base do módulo TAME foi redefinida em termos de interfaces, classes abstratas e padrões de projeto. Durante o projeto arquitetural, as classes identificadas durante a fase de Planejamento continuaram em vigor, porém o relacionamento entre elas mudou, devido ao surgimento das interfaces e das classes abstratas. Em outras palavras o relacionamento das classes identificadas anteriormente foi alterado e é feito por intermédio de interfaces e classes abstratas.

O macro *milestone* da fase de Elaboração é o projeto arquitetural, que será discutido em mais detalhes no próximo capítulo. O Capítulo 4 mostra os detalhes do projeto arquitetural do módulo TAME, focando as razões da escolha do uso de certas classes, interfaces e padrões de projeto. Os demais diagramas de classes que compõem a arquitetura do módulo TAME também são mostrados no Capítulo 4.

5.3. Fase de Construção

A fase de construção foi composta basicamente pelas atividades de Implementação e Teste, com algumas passagens pela atividade de Projeto para efetuar algumas correções no projeto arquitetural do sistema.

Durante a Implementação foi implementado o que foi obtido durante a fase de Elaboração. Como resultado dessa fase temos o código fonte referente às classes do módulo TAME, uma documentação (Javadoc) com a descrição de todas as classes, seus relacionamentos, operações e atributos e um arquivo .jar (Java Archive) contendo todos os arquivos de biblioteca necessários para o uso do módulo TAME.

Durante a atividade de Teste fizemos o planejamento dos testes com base no modelo de tarefas pertencente à arquitetura do sistema. Durante o planejamento foi definida a estratégia de teste a ser utilizada e as fontes de informação necessárias para sua execução. Como resultado de tal planejamento obtivemos um documento chamado Plano de Testes.

Após o planejamento, fizemos o projeto dos testes. É através do projeto de testes que identificamos e descrevemos os casos de teste e como tais testes devem ser executados.

Depois do projeto, os casos de teste identificados foram implementados. A implementação envolve a configuração do sistema para simular o estado desejado em cada caso. Após a implementação executamos os casos de teste e por fim, analisamos os resultados obtidos. Os detalhes a respeito dos testes realizados podem ser encontrados no Capítulo 5.

O macro *milestone* da fase de Implementação foi uma versão, ainda em teste, do módulo TAME com as funcionalidades implementadas.

5.4. Fase de Transição

Durante a fase de transição o módulo TAME foi disponibilizado para ser integrado ao módulo TAOS-Graph, e o resultado da integração dos dois módulos foi a ferramenta iTAOS.

Os artefatos que acompanham a distribuição do módulo TAME são:

- Código fonte – o módulo TAME tem seu código aberto, estando susceptível a alterações conforme seja necessário;
- Documentação das classes (API) – trata-se de um documento HTML que permite a navegação entre as diversas classes que compõem o módulo com explicações de como utilizar cada uma delas adequadamente. Esse documento é direcionado para quem deseja utilizar o TAME como plataforma para implementar uma ferramenta para análise e modelagem da tarefa baseada no formalismo TAOS.
- Arquivos de *byte code* Java – arquivos pré-compilados do código fonte do módulo TAME.

- Biblioteca (TAME.jar) – os arquivos pré-compilados do código fonte do módulo TAME compactados em único arquivo. Esse arquivo pode ser utilizado em substituição aos arquivos citados no item anterior.
- Arquivo de *script* (build.xml) – um arquivo para ser rodado com o uso da ferramenta ANT. Ele contém uma configuração para realizar automaticamente um determinado conjunto de ações referentes à compilação do código fonte, geração da documentação da API e geração do arquivo de biblioteca. A ferramenta ANT é distribuída junto ao módulo TAME e pode ser encontrada no *site*: <http://jakarta.apache.org/ant>.

A distribuição do módulo TAME é feita através do *site* do projeto iTAOS, <http://www.dsc.ufcg.edu.br/~itaos>, ou pelo CD da ferramenta iTAOS, disponível no departamento de Sistemas e Computação da UFCG. A API do módulo também pode ser consultada *on-line* a partir desse mesmo *site*.

6. Conclusão

Neste capítulo, foi visto como foi feita a escolha do processo de desenvolvimento do módulo TAME. Tal escolha teve que ser feita levando-se em consideração o desenvolvimento do módulo TAOS-Graph em um processo separado, segundo o princípio da independência do diálogo. A seguir mostramos as razões de nossa escolha e vimos também que foi necessário fazer adaptações ao processo escolhido e porque tais necessidades surgiram. Foram mostradas as adaptações feitas e, por fim, descrevemos cada fase do processo e identificamos os macro *milestones* de cada uma delas, inclusive como é feita a distribuição do módulo TAME e da ferramenta iTAOS.

Capítulo 5 – Projeto Arquitetural

1. Introdução

Esse capítulo tem foco na arquitetura do módulo TAME, contendo discussões a respeito das principais classes da arquitetura e suas principais operações e relacionamentos, meio de persistência utilizado e padrões de projeto escolhidos. A arquitetura base já foi introduzida no Capítulo 3, durante este capítulo teremos descrições mais detalhadas das decisões de projeto tomadas durante a evolução de tal arquitetura.

2. Estrutura do Sistema

Essa seção mostra a estrutura estática do módulo TAME. Mostraremos as decisões tomadas com relação ao uso de interfaces de classes e classes abstratas na arquitetura do sistema para aumentar a possibilidade de reuso e facilidade de manutenção. Note que o termo “interfaces de classes” não tem nenhum tipo de ligação com o termo interface do usuário. Logo a seguir temos uma definição do que são interfaces de classes.

2.1. Interfaces de Classes

Uma interface de classe define o contrato da classe com o código que a chama, ou que a utiliza. Dada a definição do que são interfaces de classe, ao longo deste capítulo não mais nos referiremos ao termo interface como sendo referente à interface do usuário, mas como sendo referente a interfaces de classe.

O uso de interfaces de classes em um projeto arquitetural aumenta o reuso do código cliente, ou seja, o código que manipula as classes que implementam tais interfaces, facilitando assim sua manutenção. Quando utilizamos interfaces, definimos que métodos podem ser chamados, e apenas esses métodos são chamados no código cliente, logo o cliente sempre funcionará desde que as classes utilizadas por ele respeitem sempre as mesmas interfaces.

Esse tipo de interface define a API para as classes que a implementam, ou seja, toda classe que implementar uma determinada interface deve implementar os métodos que essa interface define.

Foram geradas interfaces para cada uma das classes que compõem a arquitetura base do sistema, mostrada no capítulo anterior, Figuras 8 e 9. Identificamos ainda a interface *IGenericTask*, que surgiu para abstrair o que há de comum entre as interfaces *ITask* e *IAction*. Assim, obtivemos a hierarquia de interfaces mostrada na Figura 11, abaixo:

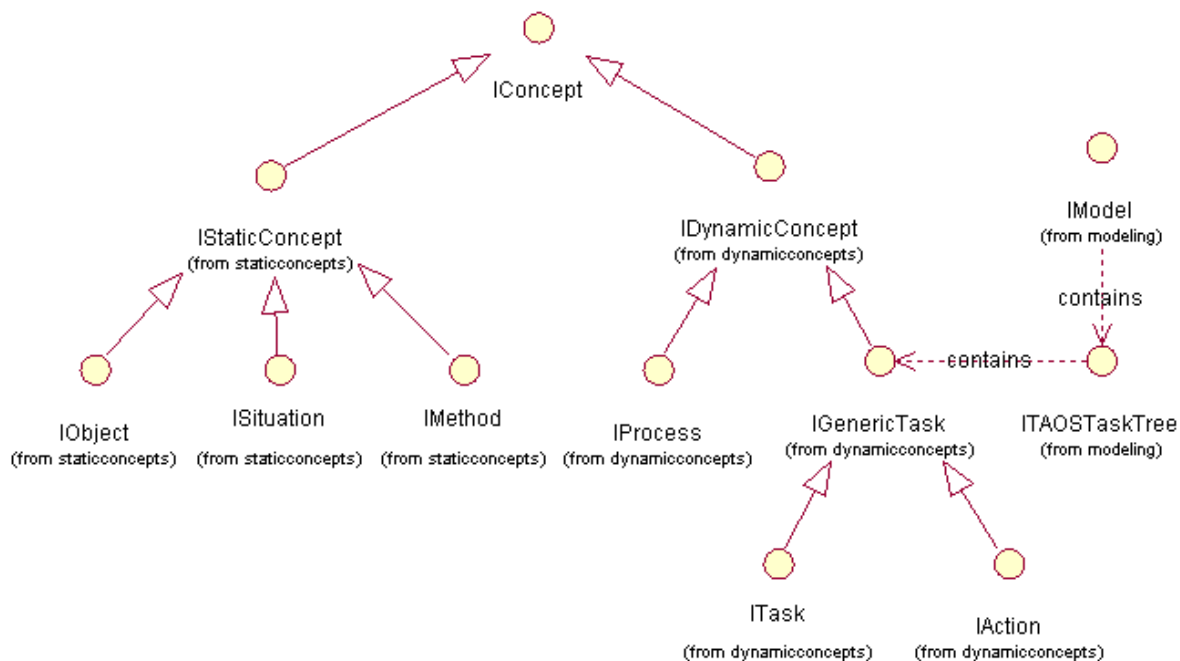


Figura 13: Hierarquia de interfaces da arquitetura base

2.2. Classes Abstratas

Se duas ou mais classes fazem parte de uma mesma hierarquia e tais classes possuem métodos que realizam o mesmo processamento, então, para facilitar a manutenção destas classes, podemos fazemos o uso de classes abstratas. E utilizando herança definimos o restante das classes que devem herdar o método da classe abstrata.

Desta maneira, se desejamos mudar a implementação de um método não é necessário fazer a alteração em cada uma das classes da hierarquia, basta fazer a alteração na classe abstrata.

A partir da arquitetura base do módulo TAME identificamos quatro classes abstratas que encapsulam processamentos em comum entre algumas classes. Essas classes são: *AbstractConcept*, *AbstractStaticConcept*, *AbstractDynamicConcept* e *AbstractGenericTask*. A classe *AbstracGenericTask* não está na arquitetura base preliminar do módulo TAME nem no meta-modelo da linguagem TAOS, porém ela foi gerada para encapsular a implementação padrão de alguns métodos em comum entre as classes *Task* e *Action* em vista de facilitar a

manutenção dessas duas classes. *AbstractGenericTask* implementa a interface *IGenericTask* identificada anteriormente. Na Figura 12 podemos observar toda a hierarquia entre as classes identificadas durante a fase de Planejamento e as classes abstratas e interfaces identificadas durante a fase de Elaboração, *workflow* de Projeto e de Implementação das demais fases.

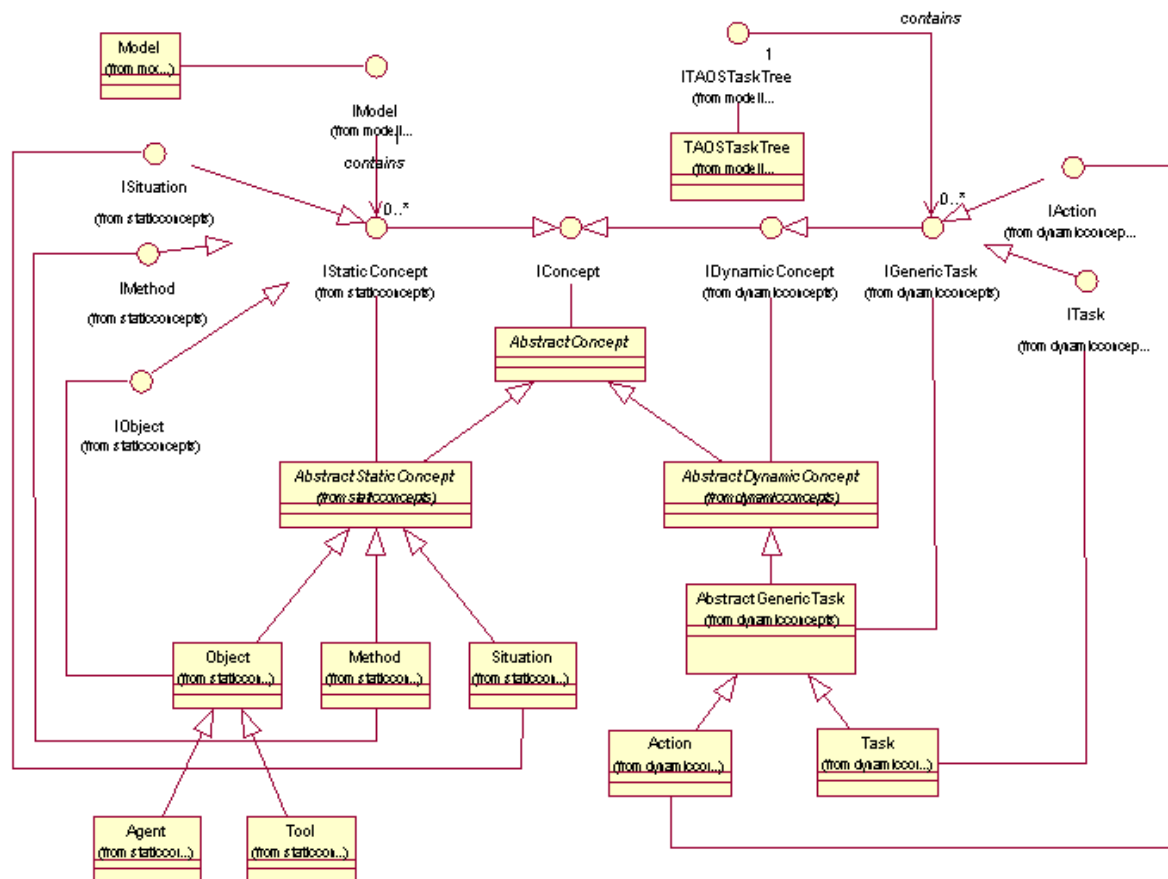


Figura 14: Hierarquia de classes abstratas da arquitetura base

2.3. Métodos e Atributos

Uma descrição detalhada de cada método e atributo de cada classe e interface pertencentes ao módulo TAME pode ser encontrada no JavaDoc que acompanha sua distribuição.

3. Comportamento Dinâmico do Sistema

O comportamento dinâmico do módulo TAME está representado no modelo de tarefas obtido durante a fase de Planejamento. O modelo de tarefas contém informações a respeito de como cada funcionalidade deve ser executada contendo o fluxo e as condições de execução de

cada uma delas. A partir do modelo também é possível observar o estado dos objetos do domínio antes e após a execução de cada tarefa (funcionalidade).

4. Persistência dos Modelos de Tarefa

A persistência dos modelos de tarefa construídos através do módulo TAME é feita em arquivos XML que respeitam o DTD disponível no Anexo D.

A persistência e a recuperação dos modelos de tarefas é feita através de JATO (Krumel, 2001a, Krumel 2001b, Krumel 2001c), que é uma API para conversão de objetos Java em arquivos XML e vice versa. Para o armazenamento, JATO interpreta o *script* de conversão de instâncias da classe *Model* para arquivos XML. E para a recuperação, JATO interpreta o *script* de conversão de arquivos XML para instâncias da classe *Model*.

4.1. JATO

JATO é uma API Java e uma linguagem XML de código aberto para transformar documentos XML em um conjunto de objetos Java e vice-versa. Os *scripts* JATO descrevem as operações a serem executadas e deixam os algoritmos de implementação das operações para um interpretador. Um *script* JATO expressa os relacionamentos entre elementos XML e objetos Java, liberando o desenvolvedor de escrever *loops* iterativos, rotinas recursivas, códigos para verificação de erros, e muitos outros códigos susceptíveis a erros, verbais e monótonos para manipulação de XML.

Jato possui muitas vantagens sobre APIs Java/XML tais como: JDOM, SAX e DOM. Entre tais vantagens podemos citar:

- JATO faz com que os projetistas Java e XML dirijam sua atenção para resolver os problemas relacionados com o domínio da aplicação e não em como fazer o mapeamento.
- Com JATO os desenvolvedores simplesmente expressam os elementos XML que devem ser mapeados para/de classes Java específicas. O interpretador JATO, então, implementa os algoritmos de *parsing* e geração de XML necessários para executar as ações desejadas.
- Utilizar XML para descrever transformações de/para XML em aplicações Java é bem natural. Quando estamos manipulando documentos XML diretamente com o código, a maior parte do trabalho é de copiar e colar.

4.2. Classes de Persistência

A classe que interage com JATO é *JatoXMLObject*. Essa classe é responsável pela persistência dos modelos de tarefa e implementa a interface *IXMLObject*.

A classe *IModel* utiliza a classe *JatoXMLObject* para fazer a persistência de suas entidades em arquivos XML e para recuperá-las posteriormente. Porém, outra classe pode ser utilizada para realizar as atividades de persistência, desde que ela também implemente a interface *IXMLObject*. Assim é possível utilizar outras estratégias de persistência que utilizem outras APIs de mapeamento Java/XML ou estratégias que utilizem ferramentas.

Para utilizar outra classe para realizar as atividades de persistência basta fazer o *override* do método `getXMLObject()` em uma subclasse de *Model* e passar a utilizar a subclasse no código cliente.

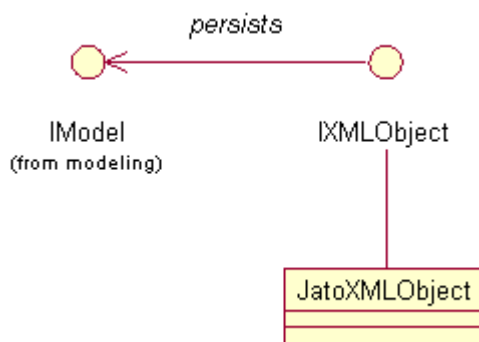


Figura 15: Classe responsável pela persistência dos modelos

A primeira coisa a ser feita para salvar um modelo é instanciar uma classe que implemente a interface *IXMLObject*, que no nosso caso é *JatoXMLObject*, da seguinte maneira:

```
IModel modelo = new Model();
```

```
IXMLObject persistenceObject = new JatoXMLObject();
```

Em seguida devemos definir o script que a classe utilizará para realizar a persistência.

Isso é feito da seguinte forma:

```
PersistenceObject.setPersistScript("nomeDoArquivoDeScript");
```

Enfim devemos executar os seguintes passos:

```
File f = new File("nomeDoArquivo");
```

```
persistenceObject.persist(modelo, new FileOutputStream("nomeDoArquivo"));
```

5. Padrões de Projeto

Como foi dito anteriormente, alguns padrões de projeto foram utilizados durante o projeto do módulo TAME para aumentar a possibilidade de reuso e facilitar a manutenção. Tais padrões podem ser vistos nesta seção.

5.1. Façade

O padrão *Façade* provê uma interface unificada para um conjunto de interfaces em um subsistema. Esse padrão define uma interface de mais alto nível, deixando o subsistema mais fácil de usar. No projeto do módulo TAME esse padrão é definido pela interface *IModel*. Através de uma classe que implemente tal interface (*Model*) é possível realizar todas as operações necessárias para manipular um modelo de tarefas.

Algumas das ações que podemos realizar, considerando o código abaixo, são:

IModel modelo = new Model();

- Adicionar conceitos ao modelo:
modelo.addConcept(conceito);
- Obter uma representação HTML do modelo:
modelo.getHTMLRepresentation();
- Obter um conceito através de seu nome:
modelo.getMethodByName("nomeMetodo");
modelo.getSituationByName("nomeSituacao");
- Remover conceitos:
modelo.removeConcept(conceito);
- Salvar o modelo em meio persistente:
modelo.saveToFile("nomeDoArquivo");

5.2. Abstract Factory

Provê uma interface para criar famílias de objetos relacionados ou interdependentes sem especificar suas classes concretas. Com o uso desse padrão é possível utilizar novas classes no código cliente sem a necessidade de alterações no mesmo, bastando mudar a classe que representa o *Abstract Factory*. No projeto do módulo TAME esse padrão é definido pela interface *ITAOSFactory* e pela classe *TAOSFactory* que a implementa. A partir de uma

instância da classe *TAOSFactory* é possível criar objetos de qualquer uma das classes que representam o meta-modelo TAOS e das classes que as manipulam: *Model* e *TAOSTaskTree*. Com o uso desse padrão é possível utilizar outras classes com poucas modificações no código cliente, desde que tais classes respeitem as interfaces definidas na arquitetura do módulo TAME. Por exemplo:

```
ITAOFactory fac = TAOSFactory.getInstance();  
IAction a = fac.createAction("nomeAcao"); // cria uma ação  
ISituation s = fac.createSituation("nomeSituacao"); // cria uma situação  
IModel m = fac.createModel("nomeDoArquivo"); // cria um modelo a partir de um  
// arquivo
```

5.3. Factory Method

O padrão *Factory Method* define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe instanciar. Esse padrão permite uma classe repassar a responsabilidade de instanciação para subclasses.

Esse padrão é utilizado na implementação das classes que compõem o padrão *Abstract Factory*.

5.4. Singleton

Esse padrão garante que uma classe tenha uma única instância e provê um ponto global de acesso à instância, o método *getInstance()*. No projeto do módulo TAME o *Singleton* é utilizado na classe *TAOSFactory*, pois normalmente uma aplicação só precisa de uma única instância de um *Abstract Factory* por família de produtos. Por exemplo:

```
ITAOFactory fac = TAOSFactory.getInstance();
```

Note que não chamamos o construtor da classe diretamente, e o método *getInstance()* mantém o controle da quantidade de instâncias da classe no sistema.

5.5. Observer

Esse padrão define uma dependência de um para muitos entre objetos, objeto observado e observadores, de forma a avisar e atualizar os observadores quando o estado do observado muda. No projeto do módulo TAME toda classe que representa um conceito do meta-modelo TAOS é uma subclasse da classe *java.util.Observable* da plataforma Java,

podendo, portanto, ser observada por uma classe que implemente a interface *java.util.Observer*, também da plataforma Java. Podemos ver o relacionamento entre observador e observado na Figura 14. Assim, qualquer que seja o código cliente, ele será sempre avisado de mudanças de estado sofridas pelos conceitos de um modelo de tarefas, desde que tal cliente implemente a interface *java.util.Observer* e esteja cadastrado como observador do conceito o qual ele tem interesse.

Em outras palavras, qualquer subclasse de *AbstractConcept* pode ser observada por um código cliente que implemente a interface *java.util.Observer*. Para que o código cliente seja capaz de receber as mensagens de atualização da instância observada é necessário que ele chame o método *addObserver(Observer o)* da instância passando ele mesmo como parâmetro. Assim, sempre que o estado da instância observada mudar, ela chama o método *update(Observable o, Object arg)* do observador, passando como parâmetros ela mesma e um objeto a ser utilizado pelo observador. No caso do projeto do módulo TAME, esse segundo parâmetro é o nome do atributo que mudou na instância observada. Por exemplo, se o atributo “name” da instância mudar então o parâmetro “arg” é a string “name”. Assim o código cliente pode tomar decisões de acordo com o que foi mudado.

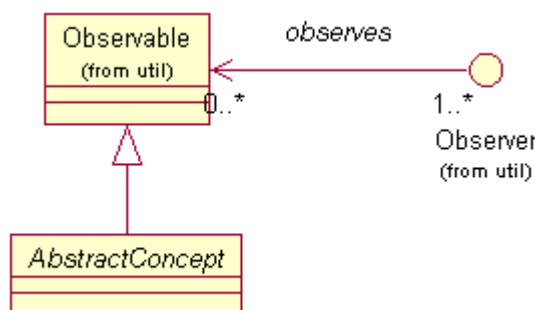


Figura 16: Padrão Observer

Esse padrão de projeto é muito útil quando utilizamos alguma forma gráfica para representar algum outro componente não visível de um sistema. Assim o componente não visível pode avisar quando seu estado mudar. Vamos considerar o código abaixo:

```

Observer componenteGrafico = new Janela();
IConcept conceito = TAOSFactory.getInstance().createAction("nomeAcao");
conceito.addObserver(componenteGrafico);
conceito.setName("nomeAcaoAlterado");
  
```

O código acima garante que quando o nome do objeto conceito mudar, o componente gráfico será avisado e assim, poderá realizar a ação adequada à situação.

5.6. Template Method

Define o esqueleto de um algoritmo numa operação, deixando que subclasses completem algumas das etapas. O padrão *Template Method* permite que subclasses redefinem determinadas etapas de um algoritmo sem alterar a estrutura do algoritmo. Esse padrão é utilizado na classe *AbstractTaskTree* no método `numerateTasks(ITask start)`, que realiza a numeração automática das tarefas da árvore de tarefas. Na classe *AbstractTaskTree* esse método é abstrato, tendo que ser implementado em suas subclasses, e é chamado sempre que uma tarefa é adicionada ou retirada da árvore. O *Template Method* foi utilizado no projeto do módulo TAME para que seja possível mudar facilmente a maneira como as tarefas são numeradas, com poucas modificações no código cliente.

5.7. Interpreter

Dada uma linguagem, esse padrão define uma representação de sua gramática e um interpretador que usa a representação da gramática para interpretar sentenças da linguagem.

O padrão *Interpreter* foi utilizado para implementar a gramática que representa o atributo *body* da classe *Method*. Cada um dos operadores da linguagem TAOS é representado por uma classe que implementa a interface *IExpression*, como pode ser visto na Figura 15, abaixo:

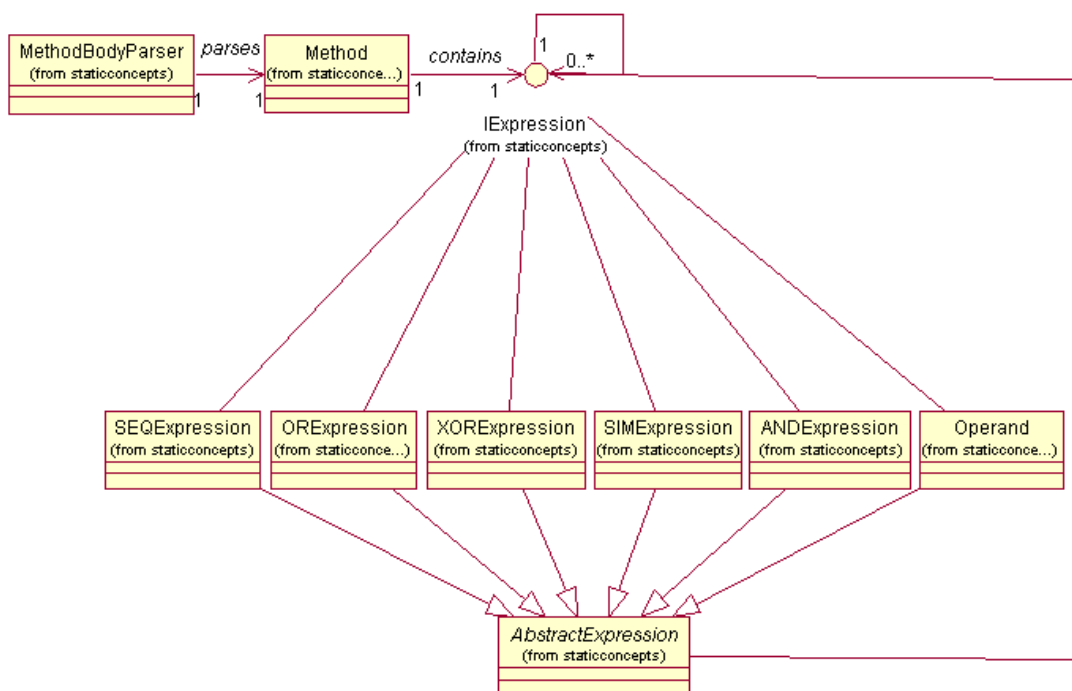


Figura 17: Classes que representam os operadores da linguagem TAOS

Toda expressão, com exceção das instâncias da classe *Operand*, pode ter zero ou mais sub-expressões ou operandos. A classe *Operand* é usada para representar as tarefas envolvidas em uma expressão.

A classe *Method* utiliza a classe *MethodBodyParser* para criar uma árvore que representa a estrutura de expressão, sub-expressões e operandos que representam o corpo do conceito método do meta-modelo TAOS através do método `parse()`. A partir dessa árvore é que a classe *Method* cria as instâncias de *IExpression* de uma maneira que estas instâncias representem a semântica do atributo *body*.

A princípio essa estrutura faz apenas a verificação de pós e pré-situações das tarefas envolvidas no conceito método (instância da classe *Method*) de uma tarefa através do método `verifySituations()`, definido na interface *IExpression* e implementado nas classes que a implementam. Em uma outra versão do módulo TAME deve ser implementado a execução das tarefas envolvidas em um conceito método através do método `execute()`, definido na interface *IExpression* e implementado na classe *AbstractExpression* para não fazer nada, servindo apenas de gancho para uma versão futura do módulo TAME.

6. Conclusão

Este capítulo mostrou o projeto arquitetural do módulo TAME. Foi possível visualizar graficamente as principais classes que compõem sua arquitetura e os principais padrões de projeto utilizados em vista de facilitar futuras alterações e sua manutenção. Algumas classes não foram vistas durante esse capítulo por não possuírem grande importância arquitetural. Uma listagem com todas as classes identificadas durante o processo de desenvolvimento pode ser vista no JavaDoc (API) que acompanha a distribuição do módulo.

É importante notar que existiu uma preocupação em definir um projeto arquitetural o mais robusto possível para que a ferramenta possa resistir a futuras mudanças facilmente e assim não deixar de ser usada.

Capítulo 6 – Descrição do Sistema

1. Introdução

Esse capítulo contém informações a respeito dos requisitos funcionais implementados pelo módulo TAME. Estes requisitos foram identificados durante a fase de Planejamento através das entrevistas feitas com futuros usuários, que resultou no modelo da tarefa “Utilizar a Ferramenta iTAOS para Analisar e Modelar Tarefas”, durante a atividade de análise e modelagem da tarefa do usuário, que compõe o *workflow* de Requisitos de nosso processo.

Vale lembrar que o módulo TAME não é uma ferramenta completa, que possui meios de interação, ele apenas fornece, através de sua API, meios para que os requisitos funcionais aqui discutidos possam ser implementados em uma ferramenta completa, a ferramenta iTAOS.

2. Identificação dos Requisitos Funcionais

Foi visto que *use cases* e tarefas têm muitas semelhanças. No Processo Unificado, processo utilizado durante o desenvolvimento do módulo TAME, *use cases* estão diretamente relacionados com os requisitos funcionais de um sistema. No nosso caso, tais requisitos foram identificados a partir das tarefas do modelo de tarefas obtido durante a fase de Planejamento.

A partir das tarefas de nosso modelo de tarefas foi possível identificar os seguintes requisitos funcionais:

1. Criar novos modelos de tarefas.
2. Salvar modelos de tarefas.
3. Recuperar modelos de tarefas existentes.
4. Editar modelos de tarefas.
5. Importar conceitos.
6. Adicionar atributos aos conceitos.
7. Imprimir modelos de tarefas.
8. Verificar completude das informações de modelos de tarefas.
9. Verificar coerência das informações de modelos de tarefas.

10. Simular modelos de tarefas.

3. Criação de Novos Modelos de Tarefas

O módulo TAME deve permitir que, através do módulo TAOS-Graph, o usuário possa criar novos modelos de tarefas quando desejado.

As classes que permitem a criação de novos modelos são: a classe *Model*, através de seu próprio construtor, e a classe *TAOSFactory*, através de um dos seus métodos de criação.

4. Salvar Modelos de Tarefas

Além da criação de modelos de tarefas, o módulo TAME permite que os modelos criados possam ser salvos em meio persistente, permitindo que o usuário possa realizar modelagens incrementais.

A classe responsável por esta funcionalidade é a classe *JatoXMLObject*, que faz a representação do modelo de tarefas em XML e salva tal representação em meio persistente. A classe *Model* possui uma instância de *JatoXMLObject* e a utiliza para fazer sua própria persistência.

5. Recuperar Modelos de Tarefas Existentes

Assim como é possível salvar modelos de tarefas em meio persistente, é possível recuperá-los. O modelo recuperado possui as mesmas informações e o mesmo estado que o modelo salvo anteriormente.

A classe responsável por esta funcionalidade é a classe *JatoXMLObject*, que transforma um arquivo XML, com as devidas informações, em um modelo de tarefas. Para fazer a recuperação de modelos, deve ser usada a classe *TAOSFactory* e um de seus métodos de criação, passando-se como parâmetro o local onde se encontra o arquivo referente ao modelo.

6. Edição de Modelos de Tarefas

O módulo TAME permite que um modelo, novo ou recuperado de arquivo persistente, possa ser manipulado através da edição de sua árvore de tarefas e dos descritores de seus conceitos.

A forma como a interação com o módulo deve ser feita para que a manipulação seja possível está fora do escopo do projeto do módulo TAME, porém, podemos assegurar, através dos testes funcionais descritos no Capítulo 6, que é possível realizar as seguintes funções:

1. Inserir tarefas na árvore de tarefas. Podendo ela ser posicionada como raiz da árvore ou como sub-tarefa de outra tarefa da árvore.
2. Excluir tarefas da árvore de tarefas.
3. Alterar o pai de uma tarefa.
4. Editar os descritores das tarefas e dos conceitos relacionados.

Vale lembrar que uma das dificuldades encontradas quando manipulamos modelos de tarefas manualmente é manter a numeração correta das tarefas da árvore. Portanto, uma das funcionalidades que o módulo TAME deve considerar é a numeração automática das tarefas.

As classes responsáveis por estas funcionalidades são: *Model* e *TAOSTaskTree*. E as classes que representam os conceitos, cujos descritores podem ser editados são: *Object*, *Agent*, *Tool*, *Situation*, *Method*, *Action* e *Task*.

7. Reuso de Conceitos

O módulo TAME também permite que conceitos pertencentes a um modelo de tarefas possam ser utilizados em outros modelos através da funcionalidade de importação de conceitos. Com essa funcionalidade é possível fazer o reuso de conceitos para economizar tempo de trabalho e/ou aproveitar idéias. Quando um conceito é importado, todos os conceitos que estão relacionados com ele também são importados para o modelo. A classe responsável por esta funcionalidade é a classe *Model*.

8. Adição de Atributos aos Conceitos.

É possível adicionar novos atributos aos descritores de qualquer conceito, conforme seja necessário. O módulo TAME permite que o usuário defina novos campos para os conceitos de um modelo. Por exemplo, além das informações que podem ser armazenadas em um agente, o usuário pode adicionar um novo campo chamado “Sexo” e definir seu valor como sendo “masculino”. Essa funcionalidade é definida pelas classes que representam o meta-modelo TAOS: *Object*, *Agent*, *Tool*, *Situation*, *Method*, *Task* e *Action*.

9. Impressão de Modelos de Tarefas

O módulo TAME não oferece a possibilidade de imprimir a árvore de tarefas, pois essa funcionalidade deve ser preocupação de quem fornece a representação gráfica de tal árvore. Porém, este módulo oferece uma facilidade para imprimir os descritores dos conceitos presentes no modelo, que é a geração de um fluxo com a representação em HTML de tais conceitos. O fluxo é gerado pela classe *Model*.

10. Verificação da Completude das Informações

O módulo TAME também se encarrega da verificação da completude das informações. Com essa funcionalidade, o sistema, quando solicitado, é capaz de identificar conceitos cujos descritores não tenham sido completamente preenchidos. Nesse caso, o módulo TAME envia uma mensagem para seu cliente (TAOS-Graph) indicando a ocorrência e localização do erro.

Essa funcionalidade é definida pelas classes que representam o meta-modelo TAOS: *Object*, *Agent*, *Tool*, *Situation*, *Method*, *Task* e *Action* e pode ser chamada a partir da classe *Model*.

11. Verificação da Coerência das Informações

Durante a verificação da coerência das informações, o módulo TAME verifica se existem informações que não façam sentido e comprometam o modelo. Por exemplo, a restrição da pré-situação de uma tarefa pode não estar de acordo com a restrição da pós-situação de outra ou então uma das tarefas envolvidas no corpo de um método não faz parte do conjunto de sub-tarefas de uma tarefa. Em qualquer caso, o módulo TAME envia uma mensagem para seu cliente (TAOS-Graph) indicando a ocorrência e localização do erro.

Essa funcionalidade é implementada pela classe *Method* e pode ser chamada a partir das classes *Task* ou *Model*.

11.1. Regras para Verificação das Situações

Cada operador (temporal ou lógico) da linguagem TAOS influencia na verificação das situações das sub-tarefas de uma tarefa. Para que as condições de execução e de término das sub-tarefas sejam coerentes com o encadeamento imposto pelo operador, o usuário deve obedecer algumas regras. Para descrever essas regras, vamos considerar que situações são

conjuntos compostos por predicados e que os operadores matemáticos utilizados na Teoria dos Conjuntos podem ser aplicados a esses conjuntos. Para um melhor entendimento das regras, vamos considerar uma estrutura de tarefas como a descrita pela Figura 16, abaixo:

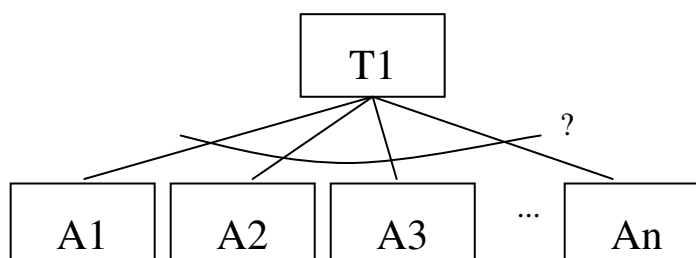


Figura 18: Estrutura de Tarefas

Vamos chamar a pré-situação de $T1$ de $PréT1$ e sua pós-situação de $PósT1$. A mesma nomenclatura será usada para as sub-tarefas. As regras que devem ser respeitadas para que as situações da tarefa e suas sub-tarefas estejam corretamente especificadas são as seguintes:

- Para sub-tarefas sob a ação do operador SEQ
 1. $PréT1 \supset PréA1$;
 2. $\bigcup_{1 \leq i \leq n} PósAi \cup PréT1 \supset PréAj$, onde $j = n + 1$;
 3. $PósAi-1 \cap PréAi \neq \{\}$, onde $2 \leq i \leq n$;
 4. $PósT1 \cap PósAn \neq \{\}$;
 5. $\bigcup_{1 \leq i \leq n} PósAi \supset PósT1$.
- Para sub-tarefas sob a ação dos operadores AND, SIM ou PAR
 1. $PréT1 \supset PréAi$, onde $1 \leq i \leq n$;
 2. $\bigcup_{1 \leq i \leq n} PósAi \supset PósT1$;
 3. $PósT1 \cap PósAi \neq \{\}$, onde $1 \leq i \leq n$.
- Para sub-tarefas sob a ação dos operadores OR ou XOR
 1. $PréT1 \supset PréAi$, onde $1 \leq i \leq n$;
 2. $PósAi \supset PósT1$, onde $1 \leq i \leq n$.

12. Simulação de Modelos de Tarefas

A simulação de modelos de tarefas foi um requisito funcional identificado, mas este requisito foi deixado para ser implementado em uma futura versão da ferramenta iTAOS, pois

durante as entrevistas realizadas na fase de Planejamento os futuros usuários deram prioridade à possibilidade de realizar a modelagem para depois ser possível realizar sua simulação.

13. Conclusão

Neste capítulo, foram descritos os requisitos funcionais que o módulo TAME deve obedecer e as principais classes que implementam cada uma das funcionalidades exigidas. Vale lembrar que o módulo TAME não oferece cada uma dessas funcionalidades propriamente dita, já que ele não é uma ferramenta completa, com interface do usuário. O que o módulo TAME oferece é uma API que pode ser utilizada para dar origem às funcionalidades discutidas ao longo deste capítulo. No capítulo seguinte, que descreve os testes funcionais, veremos que tais funcionalidades podem, de fato, serem implementadas com o uso do módulo TAME.

Capítulo 7 – Testes Funcionais

1. Introdução

Este capítulo tem como objetivo mostrar os testes funcionais realizados com o módulo TAME. Esse tipo de teste é importante para mostrar que os requisitos funcionais identificados foram implementados e implementados corretamente.

Será mostrado como foram realizadas as atividades de planejamento, projeto, implementação e execução dos testes.

2. Planejamento dos Testes

O módulo TAME não é uma ferramenta completa que contenha uma interface de interação, e sim um conjunto de classes que disponibilizam uma API, através da qual ele oferece seus serviços. Sendo assim, não é possível enxergar as funcionalidades que esse módulo oferece apenas olhando para ele. É necessário conhecer sua API e saber combinar as classes e seus métodos corretamente para implementar uma determinada funcionalidade. Em outras palavras, o módulo TAME não oferece as funcionalidades identificadas durante a fase de Planejamento prontas para serem usadas pelo usuário final, ele serve de plataforma para que tais funcionalidades sejam implementadas e o objetivo dos testes funcionais aqui realizados é justamente mostrar que a API do módulo TAME suporta a implementação de tais funcionalidades.

2.1. O Que Testar

Como o processo do módulo TAME foi baseado em tarefas, e não em *use cases*, o que foi testado foram justamente as tarefas do modelo de tarefas que tomamos como ponto de partida para o processo de desenvolvimento do módulo em questão. Uma alternativa seria fazer testes exaustivos dos fluxos que cada tarefa pode seguir, ou seja, testar todos os fluxos possíveis que podem ser observados no modelo. Porém esse tipo de teste é humanamente impraticável, e, sendo assim, foi feita uma escolha das tarefas consideradas mais importantes

do modelo, ou seja, aquelas que serviram para identificar os requisitos funcionais e fizemos os testes com alguns de seus fluxos de execução.

2.2. Recursos Utilizados

Como o módulo TAME não possui uma interface de interação própria, com a qual os testes funcionais possam ser realizados através de interações diretas, decidimos escolher um *framework* que possibilita a codificação dos testes e posteriormente a execução dos mesmos. Assim, escolhemos o JUnit, que originalmente tem o propósito de ser usado em testes de unidade, mas organizamos os testes de unidade de maneira que eles representassem uma funcionalidade. O *framework* JUnit e sua documentação podem ser encontrados no *site* <http://www.junit.org>.

Outro recurso necessário para realizar os testes é um modelo de tarefas, já que as funcionalidades a serem testadas estão relacionadas com a manipulação de modelos de tarefas. Escolhemos um modelo de tarefa existente, que representa uma tarefa do mundo real: “Gerenciar Processo de Software” (Cordeiro e Lula, 2001). Foi feita a modelagem apenas das tarefas pertencentes aos dois primeiros níveis da árvore, pois o objetivo destes testes não é produzir um modelo completo e sim mostrar que é possível construir um modelo e manipulá-lo com o uso do módulo TAME.

Sendo assim, utilizamos o JUnit para fazer toda a configuração do modelo de tarefas escolhido e para realizar algumas atividades de manipulação sobre o mesmo.

3. Projeto dos Testes

Aqui mostraremos os casos de teste escolhidos para realizar os testes funcionais. Durante o projeto dos testes realizamos as atividades listadas abaixo:

- Selecionamos as tarefas a serem testadas;
- Combinamos tarefas para formar cada caso de teste;
- Selecionamos os fluxos de execução das tarefas escolhidas a serem testados;
- Identificamos as seqüências a serem seguidas para que os casos de teste pudessem ser executados, ou seja, identificamos a dependência entre cada caso de teste;
- Definimos que resultados deveriam ser obtidos com a execução de cada caso de testes.

3.1. Casos de Teste

A primeira bateria de testes a ser realizada foi com a tarefa 1.2 “Realizar Modelagem” do modelo de tarefas construído durante o processo do módulo TAME. Essa tarefa está relacionada com todas as funcionalidades que dizem respeito às operações básicas de manipulação de um modelo de tarefas, tais como: criar novos modelos, salvar, abrir modelos existentes, editar modelos existentes, adicionar atributos adicionais aos conceitos do modelo, importar conceitos de outro modelo e imprimir. Com essas funcionalidades foram feitos seis casos de teste, que podem ser vistos Tabela 2 abaixo:

Caso de Teste	Descrição	Resultados Esperados
Caso 1	Criar um novo modelo e tentar imprimir sem realizar nenhuma edição e sem salvar.	<ul style="list-style-type: none"> • Deve ser impressa uma página com informações em branco. • Não deve ser mostrada nenhuma mensagem de erro.
Caso 2	Criar um novo modelo, salvar e imprimir sem fazer nenhum tipo de edição.	<ul style="list-style-type: none"> • Deve ser criado um arquivo em meio físico que corresponda ao modelo. • Deve ser impressa uma página em branco. • Não deve ser mostrada nenhuma mensagem de erro.
Caso 3	Criar um novo modelo, editar adequadamente com os dados do modelo da tarefa “Gerenciar Processo de Software”, salvar e imprimir.	<ul style="list-style-type: none"> • Deve ser criado um arquivo em meio físico que corresponda ao modelo. • Deve ser impressa uma página com todas as informações contidas no modelo. • Não deve ser impressa nenhuma mensagem de erro.
Caso 4	Abrir um modelo existente, editar, salvar as alterações e imprimir.	<ul style="list-style-type: none"> • As alterações devem ser salvas para o arquivo que corresponda ao modelo.

		<ul style="list-style-type: none"> • Deve ser impressa uma página com todas as informações contidas no modelo, inclusive as alterações. • Não deve ser impressa nenhuma mensagem de erro.
Caso 5	Abrir um modelo existente, adicionar atributos adicionais a alguns de seus conceitos, salvar as alterações e imprimir.	<ul style="list-style-type: none"> • Deve ser impressa uma página com todas as informações contidas no modelo, inclusive os atributos adicionais. • Não deve ser impressa nenhuma mensagem de erro.
Caso 6	Abrir um modelo existente, importar conceitos de outro modelo, salvar as alterações e imprimir.	<ul style="list-style-type: none"> • Deve ser impressa uma página com todas as informações contidas no modelo, inclusive os conceitos importados. • Não deve ser impressa nenhuma mensagem de erro.

Tabela 4: Casos de Teste

A segunda bateria de testes a ser realizada foi com a tarefa 1.3 “Verificar Modelo” do modelo de tarefas construído durante o processo do módulo TAME. Essa tarefa está relacionada com todas as funcionalidades que dizem respeito às operações de verificação do modelo, tais como: verificar a completude e a coerência das informações do modelo. Com essas funcionalidades foram feitos dois casos de teste. Os casos de teste para a tarefa 1.3 podem ser vistos na Tabela 3, abaixo:

Caso de Teste	Descrição	Resultados Esperados
Caso 7	Executar o Caso 3 e fazer a verificação da coerência das informações.	<ul style="list-style-type: none"> • Deve ser mostrada uma mensagem de erro com os erro encontrados ou uma mensagem de sucesso.
Caso 8	Executar o Caso 3 e fazer a	<ul style="list-style-type: none"> • Deve ser mostrada uma

	verificação da completude das informações.	mensagem de erro com os erros encontrados ou uma mensagem de sucesso.
--	--	---

Tabela 5: Casos de Teste (Continuação)

4. Implementação dos Testes

Como dito anteriormente, os testes foram implementados com o JUnit. Foi implementada uma classe de teste para cada caso de teste.

Para agilizar a configuração de cada caso de teste, foi criada uma classe que contém os métodos fundamentais para cada um deles. Tal classe se chama *ModelManager* e contém os seguintes métodos:

- *IModel* createModel(), cria o modelo da tarefa “Gerenciar Processo de Software”.
- void printModel(*IModel* m, *String* file), cria um arquivo HTML correspondente à impressão dos descritores dos conceitos do modelo passado como parâmetro.
- void saveModel(*IModel* m, *String* file), salva o modelo para o arquivo de nome passado como parâmetro.
- *IModel* openModel(*String* f), abre o modelo do arquivo de nome passado como parâmetro.
- void verifyInformationCompleteness(*IModel* m), verifica a completude das informações do modelo.
- void verifyInformationCoherence(*IModel* m), verifica a coerência das informações do modelo.

Para cada um dos casos de teste listados nas tabelas 3 e 4 foram criadas as seguintes classes: *TestCase1*, *TestCase2*, *TestCase3*, *TestCase4*, *TestCase5*, *TestCase6*, *TestCase7*, *TestCase8*. Cada uma destas classes faz uso dos métodos da classe *ModelManager* para criar suas respectivas configurações e realizar seus respectivos fluxos de execução.

5. Execução dos Testes

Para executar os testes existem duas opções: executar cada uma das classes referentes aos casos de teste separadamente, ou utilizar um *script* que executa todas as classes em uma *suite*. A distribuição do módulo TAME inclui *scripts* para a execução dos testes funcionais.

5.1. Resultados

Os resultados dos testes referentes aos casos 1, 2 e 3 podem ser visualizados no Anexo E, e tratam-se de arquivos em formato HTML referentes à atividade de impressão. Com esses arquivos é possível ver que foi possível criar um modelo de tarefa com o uso do módulo TAME. Os resultados referentes aos casos 4, 5 e 6 também podem ser vistos em formato HTML, porém as informações contidas neles são semelhantes às informações contidas no arquivo HTML referente ao caso de teste 3, e não foram colocadas no Anexo E..

Os resultados obtidos foram satisfatórios, foi possível implementar todos os requisitos funcionais previstos corretamente. O módulo TAME tem capacidade de servir como plataforma de implementação para a ferramenta iTAOS.

6. Conclusão

Neste capítulo foram feitos os testes funcionais para o módulo TAME, mostramos como foi feito o planejamento, projeto, implementação e execução dos mesmos. O capítulo também fornece informações de como os resultados podem ser gerados.

Capítulo 8 – Discussões e Conclusão

1. Introdução

Este trabalho discorreu, inicialmente, sobre a importância do desenvolvimento de *software* interativo, enfatizando a ausência de conceitos e atividades relacionadas à concepção de interfaces do usuário na maioria das metodologias tradicionais de desenvolvimento e a necessidade de um processo de desenvolvimento centrado no usuário (UCD). Vimos que, no contexto de UCD, a análise e modelagem da tarefa são pontos fundamentais do processo. Foi feita uma ambientação com relação ao que são ferramentas de suporte à análise de modelagem da tarefa, bem como o estado atual de algumas delas e suas principais funcionalidades. Apresentamos a linguagem TAOS como linguagem para modelagem de tarefas do usuário e mostramos sua sintaxe. Foi visto também que existem dificuldades quando os modelos de tarefas do usuário são construídos manualmente e como solução foi proposto a construção de uma ferramenta de suporte à análise e modelagem da tarefa baseada na linguagem TAOS (iTAOS), que foi concebida para ser implementada em dois módulos: TAME e TAOS-Graph. O objetivo deste trabalho de dissertação foi justamente projetar e implementar o módulo TAME, segundo o princípio da independência do diálogo. Foi mostrado o processo de desenvolvimento do módulo TAME, que é uma adaptação do Processo Unificado. Apresentamos também o projeto arquitetural realizado durante o processo de desenvolvimento, onde discutimos a respeito das principais tomadas de decisão durante o processo de desenvolvimento do módulo em questão. Em seguida, apresentamos os requisitos funcionais que foram implementados pelo módulo. E, por fim, descrevemos como foram realizados os testes funcionais para garantir que todas as funcionalidades desejadas foram implementadas e implementadas corretamente.

2. Discussões dos Resultados

Vale deixar bem claro que o objetivo deste trabalho foi projetar e implementar o módulo TAME, para ser usado como componente funcional da ferramenta iTAOS. O projeto

e a implementação do módulo foram realizados com sucesso, as funcionalidades exigidas foram implementadas e testadas, estando o módulo TAME em perfeitas condições de uso.

O projeto e implementação do módulo TAME foi fundamentado em três hipóteses, que puderam ser confirmadas em sua totalidade ou em parte, como pode ser visto abaixo:

- **H1: É possível implementar a parte funcional de um sistema interativo em um processo de *software* separado do processo de *software* de sua interface do usuário, segundo o princípio da independência do diálogo.**

Essa hipótese pôde ser parcialmente confirmada já que o processo de desenvolvimento dos dois módulos da ferramenta iTAOS não foram totalmente independentes. Para que fosse possível atingir o sucesso da implementação da ferramenta foi necessário definir uma forma de comunicação entre os processos, e essa comunicação se deu pela definição da API através da qual o módulo TAME disponibiliza seus serviços. Essa API teve que ser o mais robusta possível para que solicitações de mudanças não causassem tanto impacto no que já estava construído durante o processo de desenvolvimento. É preciso também deixar claro que houve comunicação entre os dois processos de desenvolvimento a nível de ajustes da API, à medida que fosse necessário.

- **H2: É possível deduzir os requisitos funcionais de um sistema computacional a partir da análise e modelagem da tarefa do usuário.**

Realmente, foi possível fazer a dedução dos requisitos funcionais a serem implementados pelo módulo TAME a partir do modelo de tarefas obtido durante a fase de Planejamento. Vimos que há uma grande semelhança entre *use cases* e tarefas. Sendo assim, o modelo de tarefas foi usado de maneira similar à maneira como os modelos de *use cases* são usados para capturar os requisitos funcionais de um sistema. O modelo de tarefas fornece as informações necessárias para a captura de requisitos funcionais através de um único diagrama, enquanto que o modelo de *use cases* fornece tais informações de maneira distribuída entre vários diagramas. Ambos os modelos fornecem informações a respeito do fluxo a ser seguido para que uma certa atividade seja executada (comportamento dinâmico do sistema).

- **H3: O Processo Unificado (PU) é um processo adequado para o desenvolvimento do módulo TAME.**

Esta hipótese não foi totalmente confirmada, pois de acordo com nossas duas primeiras hipóteses, o processo de desenvolvimento do módulo TAME seria baseado em tarefas, porém o PU é completamente baseado em *use cases*, não sendo possível aplicar o Processo Unificado como ele realmente deve ser aplicado. Devido a essas circunstâncias utilizamos apenas a estrutura de fases e fluxos de atividades desse processo.

3. Contribuições

A contribuição deste trabalho é o módulo TAME, que satisfaz os requisitos funcionais levantados para a ferramenta iTAOS, uma ferramenta que dá suporte computacional à análise e modelagem da tarefa baseada no formalismo TAOS, permitindo que sua implementação possa seja possível. Além de resolver uma gama de problemas relacionados com a manipulação manual de modelos de tarefas, essa ferramenta pode tornar o formalismo TAOS mais popular dentro da comunidade de projetistas de interfaces homem-computador, já que essa ferramenta oferece todo um ambiente que torna fácil a manipulação dos modelos de tarefas.

Além de contribuir para a construção da ferramenta iTAOS, podemos dizer que este trabalho pode servir como uma contribuição para a discussão sobre uma possível integração entre os processos de desenvolvimento de *software* provindos da engenharia de *software* e os processos utilizados pelos projetistas de interface do usuário.

4. Trabalhos Futuros

A partir deste trabalho temos algumas sugestões para trabalhos futuros:

1. Implementar a simulação de modelos de tarefas

A simulação de modelos de tarefas foi um dos requisitos funcionais identificados durante a fase de Planejamento do módulo TAME, porém os projetistas entrevistados deram prioridade aos requisitos considerados essenciais para que a modelagem de tarefas com suporte computacional pudesse ser realizada, para que depois os modelos obtidos pudessem ser simulados. Portanto, a funcionalidade de simulação foi deixada para um trabalho futuro.

2. Implementar uma ferramenta capaz de mapear os modelos de tarefas para modelos da interação

Uma ferramenta capaz de gerar modelos de interação a partir de modelos de tarefas seria útil para a obtenção automática ou semi-automática da interface do usuário de sistemas computacionais.

3. Criar um processo de *software* que considere o projeto da interface do usuário

Durante o desenvolvimento do módulo TAME adaptamos o Processo Unificado para que fosse também possível desenvolver o módulo TAOS-Graph a partir de uma mesma análise de requisitos. Porém, a adaptação feita foi realizada de forma *ad hoc* sem uma maior fundamentação, já que nosso objetivo era desenvolver o módulo TAME e não, criar um processo de *software*. A adaptação foi feita apenas para suprir as necessidades de nosso projeto. E como trabalho futuro pode ser feito um estudo sobre nossa adaptação para que, a partir da mesma, possa surgir um processo de *software* que considere o desenvolvimento da interface do usuário.

9. Referências Bibliográficas

BOEHM, B. IEEE COMPUTER. A spiral model of software development and enhancement. [S.I.: s.n.], n.21, p 61-72, maio de 1988.

CIBYS, W. A. *Ergonomia e Usabilidade de Software*. Labutil, Universidade Federal de Santa Catarina, Florianópolis, Brasil, 1996.

CHAPANIS, A. e BUDURKA, W. J. Specifying human computer interface requirements, *Behavior and Information Technology*, 9, 6 (1990), 479-492.

CLOYD, M. H. Designing User-Centered Web Applications in Web Time. *IEEE Software*, [S.I.: s.n.], p. 62-69, jan/fev 2001.

CORDEIRO, P. B. e LULA, B. DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO. *Modelagem da Tarefa “Gerenciar Processo de Software”*. Disponível em: <<http://www.dsc.ufpb.br/~ulrich/RelTec/RelTec.html>. Acesso em: 03 dez 2002.

DEVAMBU, P.T.; LITMAN, D.J. Plan-Based Terminological Reasoning, *Principles of Knowledge Representation and Reasoning*, Cambridge, England, 1991.

DODANI, M. H., HUGHES, C. E., MOSHELL, J. M. Separation of Powers, *Byte*, março de 1989.

DRAPER, S. W., NORMAN, D. A. Software Engineering for User Interfaces, *IEEE Trans. Softw. Eng.* SE-11, p. 252-258, 1985.

EHRICH, R. W., HARTSON, H. R. DMS – An environment for dialogue management, *Proc. Of COMPCON81* (Washington, D. C.), IEEE, New York, p. 121, setembro de 1981.

GAMBOA, F. R.; SCAPIN, D. L. Editing MAD* task descriptions for specifying user interfaces, at both semantic and presentation levels. *INTERNATIONAL EUROGRAPHICS WORKSHOP ON DESIGN, SPECIFICATION, AND VERIFICATION OF INTERACTIVE SYSTEMS*, 4, Proceedings DSV-IS 97, Granada – Espanha: Springer-Verlag, 1997.

GOLDFARB, C. F. e PRESCOD, P. *The XML Handbook, second edition, The Charles F. Goldfarb Series on Open Information Management*. [S.I.: s.n.], 2000.

HAMMOUCHE, H. *De la modélisation des tâches utilisateurs au prototype de l'interface homme-machine*. Tese (Doutorado em Informática) – l'Université Paris VI. França: l'Université Paris VI, 1995.

HEEMANN, V. CURSO DE ERGONOMIA EM SISTEMAS DE INFORMAÇÃO, Campina Grande, Brasil, Julho de 1997.

HEINSOHN, J., KUDENKO, D., NEBEL, B., PROFITLICH, H. J. RAT – Representation of Actions in Description Logics. WORKSHOP ON TAXONOMICS REASONING. DFKI, Saarbrücken: [s.n.], 1992.

HUMPHREY, W. *A Discipline for Software Engineering*. [S.I.]: Addison-Wesley, 1995.

HUNT, J. *The Unified Process for Practitioners – Object Oriented Design, UML and Java*. Practitioner Series, 2001.

JACOBSON, I., BOOCH G. e RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, Reading, MA. 1999.

JACOBSON, I. THE RATIONAL EDGE. *Use Cases – Yesterday, Today, and Tomorrow* <<http://www.therationaledge.com>>. Acesso em: 17 abr 2003.

JOHNSON, P., JOHNSON, H., WADDINGTON R. and SHOULS A. Task-Related Knowledge Structures: Analysis, Modelling and Application. Queen Mary College, University of London, 1988.

KAFURE, I. M. *Validação do Formalismo TAOS para a Concepção de Interfaces Homem-Computador*. Dissertação (Mestrado em Informática) – Coordenação de Pós-Graduação em Informática. Campina Grande: Universidade Federal da Paraíba, 2000.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*, 2. ed. Addison-Wesley Pub, 2000.

KRUCHTEN, P. THE RATIONAL EDGE. *Planning an Iterative Project*. Disponível em: <<http://www.therationaledge.com>>. Acesso em: 14 out 2002.

KRUMEL, A. JAVA WORLD. *JATO: A new kid on the open source block, Part 1 – A new library for converting between Java and XML*. Disponível em: <<http://www.javaworld.com/javaworld/jw-03-2001/jw-0316-jato.html>>. Acesso em: 02 jun 2002.

KRUMEL, A. JAVA WORLD. *JATO: A new kid on the open source block, Part 2 – Look in-depth at Java-to-XML translation*. Disponível em: <<http://www.javaworld.com/javaworld/jw-04-2001/jw-0413-jato2.html>>. Acesso em: 02 jun 2002.

KRUMEL, A. JAVA WORLD. *JATO: A new kid on the open source block, Part 3 – Translate XML documents into Java Objects*. Disponível em: <<http://www.javaworld.com/javaworld/jw-05-2001/jw-0525-jato3.html>>. Acesso em: 02 jun 2002.

LARMAN, C. *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and the Unified Process*. 2. ed. [S.I.]: Prentice Hall PTR, 2002.

LEHRER, N. Knowledge Representation Specification Language (KRSL), ISX Corporation , 1993.

LIMBOURG, Q., PRIBEANU, C., VANDERDONCKT, J. Towards Uniformed Task Models in a Model-Based Approach. *GIST Technical Report G2001-1*. Glasgow – Escócia: [s.n.], 2001.

MEDEIROS, H. e ROUSSELOT F. Un Outil D'Aide à la Modélisation de Concepts Dynamiques: Le Système TAME. JOURNÉES ACQUISITION – VALIDATION – APPRENTISSAGE. *JAVA '95, 04/95*. Grenoble – França: [s.n.], 1995.

MEDEIROS, H. Contribution à la Construction d'Une Ontologie du Domaine Axée Sur les Actions: l'Approche TAME. Departamento de Sistemas e Computação, Univerdidade Federal da Paraíba, Campina Grande, Brasil, 1998.

MEDEIROS, H. TAOS-Graph: Uma Interface Gráfica de Frames Representando os Conceitos Estáticos e Dinâmicos de um Domínio. Departamento de Sistemas e Computação, Univerdidade Federal da Paraíba, Campina Grande, Brasil, 1998.

MEDEIROS, H., KAFURE, I. M e LULA, B. Jr. TAOS: a Task-and Action Oriented Framework for User's Task Analysis in the Context of Human-Computer Interfaces. INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, 20. *Proceedings of SCCC 2000*. Santiago – Chile: [s.n.], 2000.

MEDEIROS, F. P. A., CORDEIRO, P. B. e LULA, B. J. UNIVERSIDADE FEDERAL DE CAMPINA GRANDE – DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO. *Projeto iTAOS – Modelagem da tarefa e Fase de planejamento*. Disponível em: <http://www.dsc.ufcg.edu.br/~itaos>>. Acesso em: 20 dez 2002.

MEDEIROS, F. P., CORDEIRO, P. B., LULA, B. J. *iTAOS: a Graphical Tool to Support User's Task Description in UI Design Context*. In: V Symposium on Human Factors Computer Systems. Fortaleza: BNDE, p. 376-379, 2002.

MEDEIROS, F. P., CORDEIRO, P. B., LULA, B. J. *A graphical tool to support task description using TAOS formalism for UI design*. In: 7th ERCIM Workshop. Paris: ERCIM, p. 45-51, 2002.

MINSKY, M. *A Framework for Representing Knowledge. The Psychology of Computer Vision*. New York: McGraw-Hill, 1975.

PATERNÒ, F., MANCINI, C. & MENICONI, S. (1997), ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, in S. Howard, J. Hammond & G. Lindegaard, eds, 'Proceedings of INTERACT '97', Chapman & Hall, Sydney, pp. 362–369.

PATERNÒ, F. (1999), *Model-Based Design and Evaluation of Interactive Applications*, Springer Verlag.

PREECE, J. e ROMBACH, D. INTERNATIONAL JOURNAL OF HUMAN-COMPUTER STUDIES. A taxonomy for combining Software Engineer (SE) and Human-Computer Interaction (HCI) measurement approaches: Towards a common framework. [S.I.: s.n.], n. 41, p. 553-583, 1994.

SACERDOTI, E. D., *Planning in a Hierarchy of Abstraction Spaces*, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, U.S.A., 1974.

SACERDOTI, E. D. *The Non Linear Nature of Plans*. Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, U.S.A., 1975.

SCAPIN, D. L., PIERRET-GOLKBREICH e CHRISTINE. *Towards a Method for Task Description: MAD*, Unité de Recherche, INRIA, Rocquencourt, France, 1989.

SEBILLOTE, S. *Methodology Guide to Task Analysis with the Goal of Extracting Relevant Characteristics for Human-Computer Interfaces*. *International Journal of Human-Computer Interaction*, Le Chesnay Cedex – France, 1995.

SHNEIDERMAN, B. *Designing the User Interface – Strategies for Effective Human-Computer Interacton*. p. 97-104

SUTCLIFFE, A. G. e MCDERMOTT, M. INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES. Integrating methods of human-computer interface design with structured systems development. [S. I.: s.n.], n. 34, p. 631-656, 1991.

van WELIE, M., van der VEER, G. & ELIËNS, A. (1998a), *An Ontology for Task World Models*, in ‘5th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS98’, Abingdon, UK, pp. 57–70.

van WELIE, M., van der VEER, G. & ELIËNS, A. (1998b), *Euterpe - Tool support for analyzing cooperative environments*, in ‘Ninth European Conference on Cognitive Ergonomics’, Limerick, Ireland, pp. 25–30.

van WELIE, M. *Task-Based User Interface Design*. Tese (Doutorado em Informática) – VRIJE UNIVERSITEIT, Amsterdam, Alemanha, 2001.

VLISSIDES, J., HELM, R., GAMMA, E., JOHNSON, R. *Design Patterns – Elements of Reusable Object-Oriented Software*. [S.I.]: Addison Wesley Pub, 1994.

Anexo A: Documento de Visão

iTAOS
Documento de Visão
Versão <1.0>

Histórico de Revisão

Data	Versão	Descrição	Autor
19/06/2002		Atualização das funcionalidades a serem implementadas. Foi adicionada a funcionalidade de reutilização de conceitos pertencentes a modelos já existentes.	Pedro Barbosa Cordeiro
07/07/2002		Alterações para organizar o documento e torná-lo mais bem elaborado.	Pedro Barbosa Cordeiro

Conteúdo

1.	Introdução	77
1.1	PROPÓSITO	77
1.2	ESCOPO	77
1.3	DEFINIÇÕES, ACRÔNIMOS E ABREVIACÕES	77
1.4	REFERÊNCIAS	77
1.5	VISÃO GERAL	77
2.	Posicionamento	78
2.1.	OPORTUNIDADE DE NEGÓCIO	78
2.2.	O PROBLEMA	78
2.3	POSIÇÃO DO PRODUTO	78
3.	Perfil do Usuário	79
4.	Visão Geral do Produto	79
4.1.	PERSPECTIVA DO PRODUTO	79
4.2.	LICENÇA E INSTALAÇÃO	79
5.	Características do Produto	79
5.1	MODELAGEM DE TAREFAS	79
5.2	CRIAÇÃO DE NOVOS MODELOS	79
5.3	EDIÇÃO DE MODELOS EXISTENTES	79
5.4	EDIÇÃO DE DESCRITORES DE CONCEITOS	79
5.5	GRAVAÇÃO EM MEIO PERSISTENTE	79
5.6	RECUPERAÇÃO DE MODELOS GRAVADOS EM MEIO PERSISTENTE	80
5.7	IMPRESSÃO DE ÁRVORES DE TAREFA	80
5.8	IMPRESSÃO DE DESCRITORES	80
5.9	VERIFICAÇÃO DO CORPO DOS MÉTODOS	80
5.10	VERIFICAÇÃO DA COMPLETUDE DAS INFORMAÇÕES	80
5.11	SIMULAÇÃO	80
5.12	DEFINIÇÃO DE NOVOS ATRIBUTOS	80
5.13	REUTILIZAÇÃO DE CONCEITOS	80
6	Escala de Qualidade	80

Documento de Visão

1. Introdução

O propósito desse documento é coletar, analisar e definir necessidades e características de alto nível da ferramenta iTAOS. Ele tem foco sobre as necessidades do usuário, e porque tais necessidades existem.

1.1 Propósito

Aqui nosso propósito é dar uma visão do que é realmente a ferramenta iTAOS, e quais suas principais características.

1.2 Escopo

Este documento é relativo à ferramenta iTAOS, que será projetada e implementada em duas dissertações de mestrado da COPIN – Coordenação de Pós-Graduação em Informática, encontrada na Universidade Federal de Campina Grande (UFCG), Campina Grande – PB.

1.3. Definições, Acrônimos e Abreviações

Tais informações encontram-se no Glossário criado durante a atividade de análise e modelagem da tarefa base para o projeto e implementação da ferramenta.

1.4 Referências

- MEDEIROS, H. Contribution à la Construction d'Une Ontologie du Domaine Axée Sur les Actions: l'Approche TAME. Departamento de Sistemas e Computação, Universidade Federal da Paraíba, 1998
- MEDEIROS, H. TAOS-Graph: Uma Interface Gráfica de Frames Representando os Conceitos Estáticos e Dinâmicos de um Domínio. Departamento de Sistemas e Computação, Universidade Federal da Paraíba, 1998
- KAFURE, I. M. Validação do Formalismo TAOS para a Concepção de Interfaces Homem-Computador, Dissertação de Mestrado - COPIN, Universidade Federal da Paraíba, Campina Grande, PB, 2000

1.5 Visão Geral

O restante do documento contém informações a respeito das características dos futuros usuários do produto, das características do próprio produto.

2. Posicionamento

2.1. Oportunidade de Negócio

A ferramenta a ser construída não tem o objetivo de ser lançada no mercado de softwares. Trata-se de um software acadêmico, que tem como objetivo solucionar problemas enfrentados pelo pessoal do grupo de interfaces homem-máquina da UFCG, que utilizam o formalismo TAOS para modelar tarefas do usuário.

2.2 O Problema

Problema	Dificuldade de manipular modelos de tarefas do usuário sem auxílio de uma ferramenta computacional.
Afetados	Projetistas de interface que utilizam o formalismo TAOS para projetarem as interfaces do usuário.
Solução	Implementação de uma ferramenta computacional capaz de auxiliar no processo de análise e modelagem da tarefa do usuário. A ferramenta deve disponibilizar recursos para manipulação gráfica de árvores, assim como a manipulação das informações contidas no modelo, garantindo a completude e a coerência das mesmas.

2.3 Posição do Produto

Para	COPIN – Coordenação de Pós-Graduação em Informática da Universidade Federal da Paraíba – CAMPUS II
ITAOS	É uma ferramenta capaz de auxiliar no processo de análise e modelagem de tarefas do usuário, em vista de uma concepção da interface do usuário.

3. Perfil do Usuário

Os usuários da ferramenta são projetistas de interface que utilizam o formalismo TAOS como formalismo para modelagem da tarefa do usuário. Mais precisamente projetistas de interface do grupo de interfaces homem-máquina da UFCG e alunos de disciplinas de interfaces homem-máquina.

4. Visão Geral do Produto

4.1 Perspectiva do Produto

A ferramenta é um *software standalone*, ou seja, ela vai rodar localmente em estações de trabalho. Ela é composta por dois módulos: o TAME, resultado de um trabalho de dissertação de mestrado e o TAOS-Graph, resultado de outro trabalho de dissertação de mestrado. O primeiro módulo (TAME) é encarregado pelas funcionalidades do iTAOS, enquanto que o outro vai fazer a interface com o usuário, e terá todo um processo separado durante seu desenvolvimento.

4.2 Licença e Instalação

A ferramenta tem propósitos acadêmicos.

5. Características do Produto

5.1. Modelagem de Tarefas

A ferramenta iTAOS deve permitir que o projetista de interfaces modele tarefas do usuário.

5.2. Criação de Novos Modelos

O projetista é capaz de criar modelos de tarefas do usuário.

5.3. Edição de Modelos Existentes

O projetista é capaz de abrir modelos de tarefa existentes para edição, podendo as alterações serem salvas a qualquer momento.

5.4. Edição de Descritores de Conceitos

A ferramenta permite que o projetista de interfaces preencha as informações dos conceitos do modelo.

5.5. Gravação em Meio Persistente

O projetista de interfaces pode salvar, em meio persistente, modelos de tarefa e alterações feitas nos mesmos.

5.6. Recuperação de Modelos Gravados em Meio Persistente

O projetista de interfaces é capaz de abrir modelos de tarefa para alteração ou verificação.

5.7. Impressão de Árvores de Tarefa

A ferramenta permite a impressão das árvores de tarefa dos modelos.

5.8. Impressão de Descritores

A ferramenta permite a impressão dos descritores dos conceitos do modelo.

5.9. Verificação do Corpo dos Métodos

A ferramenta verifica se a construção do corpo dos métodos está correta, verificando a compatibilidade entre os operadores e os operandos da expressão que forma o método.

5.10. Verificação da Completude das Informações

A ferramenta faz verificações de completude e consistência das informações, garantindo que todas as informações necessárias para uma boa modelagem tenham sido preenchidas e que elas sejam consistentes.

5.11. Simulação

O modelo da tarefa do usuário pode ser executado.

5.12. Definição de Novos Atributos

O usuário é capaz de adicionar novos atributos aos conceitos do modelo, de acordo com suas necessidades.

5.13. Reutilização de Conceitos

O usuário pode utilizar conceitos de modelos já existentes.

6. Escala de Qualidade

Uma das exigências para a ferramenta é uma interface do usuário o mais intuitiva possível. Essa exigência é tão importante que a interface do usuário (TAOS-Graph) terá um processo separado para seu desenvolvimento.

Anexo B: Plano de Gerenciamento de Requisitos

iTAOS
Plano de Gerenciamento de Requisitos
Versão <1.0>

Histórico de Revisão

Data	Versão	Descrição	Autor

Conteúdo

1. Introdução	85
1.1 PROPÓSITO	85
1.2 ESCOPO	85
1.3 DEFINIÇÕES, ACRÔNIMOS E ABREVIACÕES	85
1.4 REFERÊNCIAS	85
1.5 VISÃO GERAL	85
2. Gerenciamento de Requisitos	85
2.1 ORGANIZAÇÃO, RESPONSABILIDADES E INTERFACES	85
1.1.1 <i>Usuário</i>	86
2.1.2 <i>Equipe</i>	86
2.2 TABELA DE CONTATO	86
2.3 FERRAMENTAS, AMBIENTE E INFRAESTRUTURA	86
3. Artefatos dos Requisitos	86
3.1 DESCRIÇÃO DOS ARTEFATOS	86
4. Gerenciamento de Mudanças de Requisitos	87
4.1 PROCESSAMENTO E APROVAÇÃO DE REQUISIÇÃO DE MUDANÇA	87

Plano de Gerenciamento de Requisitos

1. Introdução

Esse documento descreve guias usados no projeto para estabelecer documentos de requisitos padrões. Ele define uma estratégia geral para o mapeamento dos requisitos.

1.1 Propósito

O propósito desse plano é estabelecer e documentar uma metodologia sistemática para extrair, organizar, e documentar os requisitos do sistema.

1.2 Escopo

Esse plano provê guias para o gerenciamento do projeto iTAOS.

1.3 Definições, Acrônimos e Abreviações

Veja o Glossário para maiores informações.

1.4 Referências

Kruchten, Philippe. 1999. *The Rational Unified Process*. Menlo Park, CA: Addison Wesley Rational Unified Process®, Version 2002.05.00. Copyright © 1987 – 2001. Rational Software Corporation

1.5 Visão Geral

Esse documento contém detalhes específicos e estratégias para gerenciar os requisitos do projeto iTAOS. O documento detalha como os requisitos são organizados e administrados dentro do projeto. Ele também descreve como os requisitos serão identificados, quais seus atributos, como será o rastreamento e suas modificações.

O documento descreve o processo de gerenciamento para os requisitos, descreve os fluxos de trabalho e atividades associadas com o controle de manutenção dos mesmos e especifica os milestones a serem alcançados e, ainda define padrões a serem aderidos.

2. Gerenciamento de Requisitos

2.1 Organização, Responsabilidades e Interfaces

2.1.1 Usuário

Pessoal da área de interfaces homem-máquina da Universidade Federal de Campina Grande.

2.1.2 Equipe

Todas as tarefas serão executadas pelos alunos Pedro Barbosa Cordeiro e Francisco Petrônio Alencar de Medeiros, ambos do curso de Mestrado em Informática da Universidade Federal da Paraíba.

2.2 Tabela de Contato

Nome	Título	Organização	Contato
Pedro Barbosa Cordeiro	Mestrando	COPIN-UFCG	pedro@dsc.ufcg.edu.br
Francisco Petrônio Alencar de Medeiros	Mestrando	COPIN-UFCG	medeiros@dsc.ufcg.edu.br
Bernardo Lula Júnior	Doutor	COPOIN-UFCG	lula@dsc.ufcg.edu.br

2.3 Ferramentas, Ambiente e Infraestrutura

Ferramenta	Descrição	URL
Rational RequisitePro	Para gerenciar requisitos.	www.rational.com
Jdk1.4.0	Para geração do código	java.sun.com

3. Artefatos dos Requisitos

3.1 Descrição dos Artefatos

Tipo de Documento	Descrição
Documento de Visão	Condições ou capacidades da ferramenta iTAOS.
Glossário	Usado para capturar um vocabulário comum.
Plano de Gerenciamento de Requisitos	This document type describes requirements and strategies specific to the management and development of the Requirements Management Plan.

4. Gerenciamento de Mudanças de Requisitos

4.1. Processamento e Aprovação de Requisição de Mudança

A equipe de desenvolvimento está sempre em contato com o usuário. As mudanças de requisitos, bem como suas aprovações e soluções são apresentadas em reuniões semanais.

Anexo C: Glossário

iTAOS
Glossário
Versão <1.0>

Histórico de Revisão

Data	Versão	Descrição	Autor
03/04/2002		Inclusão de novos termos	Pedro Barbosa Cordeiro

Conteúdo

1. Introdução	92
1.1 PROPÓSITO	92
1.2 ESCOPO	92
1.3 REFERÊNCIAS	92
1.4 VISÃO GERAL	92
2. Definições	92
2.1. AÇÃO	92
2.2. AGENTE	92
2.3. ÁRVORE DE TAREFAS	93
2.4. CONCEITO	93
2.5. CONCEITO DINÂMICO	93
2.6. CONCEITO ESTÁTICO	93
2.7. DESCRITOR	93
2.8. INSTRUMENTO	93
2.9. ÍTAOS	93
2.10. MÉTODO	93
2.11. MODELO DE TAREFAS	93
2.12. OBJETO	94
2.13. OPERADOR	94
2.14. PLANO	94
2.15. PROCESSO	94
2.16. SITUAÇÃO	94
2.17. TAME (TASK ACTION MODELING ENVIRONMENT)	94
2.18. TAOS (TASK AND ACTION ORIENTED SYSTEM)	94
2.19. TAOS-GRAPH	94

Glossário

1. Introdução

Esse documento define um vocabulário de termos comuns para o projeto da ferramenta iTAOS.

1.1 Propósito

Esse documento tem o propósito de familiarizar os membros da equipe de desenvolvimento com o jargão utilizado pelos projetistas de interface, em particular, os que utilizam o formalismo TAOS.

1.2 Escopo

A lista de termos presentes nesse glossário é referente à ferramenta iTAOS,.

1.3 Referências

- MEDEIROS, H. Contribution à la Construction d'Une Ontologie du Domaine Axée Sur les Actions: l'Approche TAME. Departamento de Sistemas e Computação, Universidade Federal da Paraíba, 1998
- MEDEIROS, H. TAOS-Graph: Uma Interface Gráfica de Frames Representando os Conceitos Estáticos e Dinâmicos de um Domínio. Departamento de Sistemas e Computação, Universidade Federal da Paraíba, 1998
- KAFURE, I. M. Validação do Formalismo TAOS para a Concepção de Interfaces Homem-Computador, Dissertação de Mestrado - COPIN, Universidade Federal da Paraíba, Campina Grande, PB, 2000.

1.4 Visão Geral

A seguir temos uma lista de termos comuns ao projeto e suas respectivas definições.

2. Definições

2.1. Ação

É um conceito dinâmico. As ações são as tarefas elementares (que não podem ser decompostas). Representam as folhas de uma árvore de tarefas.

2.2. Agente

É uma entidade habilitada a executar ações.

2.3. Árvore de Tarefas

Estrutura utilizada para representar a estrutura hierárquica de tarefas de um modelo. Essa estrutura é equivalente ao conhecimento que o usuário tem a respeito da tarefa, segundo o paradigma da planificação hierárquica.

2.4. Conceito

Representa uma entidade real ou abstrata do domínio modelado.

2.5. Conceito Dinâmico

Representa entidades que possuem comportamento dinâmico, ou seja, que mudam de estado durante um intervalo de tempo considerado.

2.6. Conceito Estático

Representa entidades do domínio que possuem comportamento estático, cujo estado não muda dentro de um intervalo de tempo considerado.

2.7. Descritor

Descreve os conceitos. São as informações a respeito de um conceito.

2.8. Instrumento

Descreve as ferramentas utilizadas pelos agentes para realizar suas ações. Os agentes lidam com instrumentos que permitem a execução das ações.

2.9. iTAOS

Ferramenta para análise e modelagem de tarefas em vista da concepção da interface do usuário, baseada no formalismo TAOS.

2.10. Método

É um conceito estático que descreve os planos e ações necessárias para realizar a tarefa.

2.11. Modelo de Tarefas

Representação do conhecimento que o usuário tem a respeito de sua tarefa. Um modelo de tarefas é composto pelos conceitos definidos pelo formalismo utilizado para construí-lo.

2.12. Objeto

É um conceito estático e faz referência às entidades envolvidas nas ações. Esse conceito pode ser especializado em relação ao domínio modelado.

2.13. Operador

Mecanismo que estabelece relações temporais e/ou lógicas entre os sub-planos ou ações que compõem um plano.

2.14. Plano

É um conceito dinâmico. O plano é uma estrutura que pode ser decomposta em sub-planos e ações.

2.15. Processo

É um conceito dinâmico. O processo é o conjunto de situações observadas em intervalos de tempo diferentes. O processo pode acompanhar o histórico de uma tarefa.

2.16. Situação

É um conceito estático. Uma situação é uma estrutura que referencia o conjunto de objetos que a tarefa precisa para ser realizada, e as restrições aplicadas aos mesmos.

2.17. TAME (Task Action Modeling Environment)

Representa a semântica da linguagem definida pelo formalismo TAOS. Dentro da ferramenta iTAOS, o módulo TAME representa a parte funcional.

2.18. TAOS (Task and Action Oriented System)

Formalismo de aquisição e representação do conhecimento baseado na modelagem do domínio, validado como formalismo para análise da tarefa segundo o formalismo MAD.

2.19. TAOS-Graph

É um módulo da ferramenta iTAOS responsável pela interação com o projetista de interface que utiliza essa ferramenta.

Anexo D: DTD (Document Type Definition)

```

<!ELEMENT model (taskTree, objects, agents, tools, situations, methods)>
<!ATTLIST model
  name CDATA #REQUIRED
  description CDATA #REQUIRED
>
<!ELEMENT taskTree (plan)>
<!-- Dynamic concepts -->
<!ELEMENT plan (howToPerform, (plan | action)*, additionalAttribute*)>
<!ATTLIST plan
  number CDATA #REQUIRED
  name CDATA #REQUIRED
  description CDATA #REQUIRED
  duration CDATA #REQUIRED
  frequency CDATA #REQUIRED
  preSituation CDATA #IMPLIED
  postSituation CDATA #IMPLIED
  importance CDATA #REQUIRED
  interruptability CDATA #REQUIRED
  modality CDATA #REQUIRED
  occurrence CDATA #REQUIRED
  type CDATA #REQUIRED
  priority CDATA #REQUIRED
>
<!ELEMENT action (actionAgents, actionTools, additionalAttribute*)>
<!ATTLIST action
  number CDATA #REQUIRED
  name CDATA #REQUIRED
  description CDATA #REQUIRED
  duration CDATA #REQUIRED
  frequency CDATA #REQUIRED
  preSituation CDATA #IMPLIED
  postSituation CDATA #IMPLIED
  importance CDATA #REQUIRED
  interruptability CDATA #REQUIRED
  modality CDATA #REQUIRED
  occurrence CDATA #REQUIRED
  type CDATA #REQUIRED
  priority CDATA #REQUIRED
  status CDATA #REQUIRED
>
<!-- Static concepts -->
<!ELEMENT object (components, additionalAttribute*)>
<!ATTLIST object
  name CDATA #REQUIRED
  description CDATA #REQUIRED
  instant CDATA #REQUIRED
>
<!ELEMENT agent (components, competences, additionalAttribute*)>
<!ATTLIST agent
  name CDATA #REQUIRED
  description CDATA #REQUIRED
  instant CDATA #REQUIRED
  systemExperience CDATA #REQUIRED
  taskExperience CDATA #REQUIRED
>
<!ELEMENT tool (components, utilities, operators, additionalAttribute*)>
<!ATTLIST tool
  name CDATA #REQUIRED
  description CDATA #REQUIRED

```

```

        instant CDATA #REQUIRED
    >
    <!ELEMENT situation (situationObjects, howToObtain, additionalAttribute*)>
    <!ATTLIST situation
        name CDATA #REQUIRED
        description CDATA #REQUIRED
        instant CDATA #REQUIRED
        restriction CDATA #REQUIRED
    >
    <!ELEMENT method (additionalAttribute*)>
    <!ATTLIST method
        name CDATA #REQUIRED
        description CDATA #REQUIRED
        instant CDATA #REQUIRED
        body CDATA #REQUIRED
    >
    <!-- Remainder elements -->
    <!ELEMENT components (reference*)>
    <!-- Objects that would compound an object -->
    <!ELEMENT competences (reference*)>
    <!-- Actions that an agent would Perform -->
    <!ELEMENT utilities (reference*)>
    <!-- Actions in wich a tool would be used -->
    <!ELEMENT operators (reference*)>
    <!-- Agents that would use a tool to Perform an action -->
    <!ELEMENT howToObtain (reference*)>
    <!-- Actions that lead to a situation -->
    <!ELEMENT howToPerform (reference*)>
    <!-- Methods that tells how a plan would be Performed -->
    <!ELEMENT actionTools (reference*)>
    <!ELEMENT actionAgents (reference*)>
    <!ELEMENT situationObjects (reference*)>
    <!ELEMENT additionalAttribute EMPTY>
    <!ATTLIST additionalAttribute
        name CDATA #REQUIRED
        value CDATA #REQUIRED
    >
    <!ELEMENT reference EMPTY>
    <!ATTLIST reference
        name CDATA #REQUIRED
    >
    <!ELEMENT objects (object*)>
    <!ELEMENT agents (agent*)>
    <!ELEMENT tools (tool*)>
    <!ELEMENT situations (situation*)>
    <!ELEMENT methods (method*)>

```

Anexo E: Resultados dos Testes

Model Name: Test Case 1 Model

Description: Model used in test case 1

Tasks

Objects

Agents

Tools

Situations

Methods

Model Name: Test Case 2 Model

Description: Model used in test case 2

Tasks

Objects

Agents

Tools

Situations

Methods

Model Name: Test Case 3 Model

Description: Model used in test case 3

Tasks

Action	
Number	1.1.1
Name	Formular Escopo do Projeto
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Formular Escopo do Projeto

Action	
Number	1.1.2
Name	Avaliar Soluções
Description	O gerente de projeto avalia as soluções apresentadas por sua equipe
Duration	0

Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Soluções

Action	
Number	1.1.3
Name	Tomar Decisões de Compra e Construção
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	

	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Tomar Decisões de Compra e Construção

Action	
Number	1.1.4
Name	Planejar Iterações
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Planejar Iterações

Action	
---------------	--

Number	1.1.5
Name	Planejar Business Case
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Planejar Business Case

Task	
Number	1.1
Name	Gerenciar Fase de Planejamento
Description	Gerente de projeto gerencia a fase de Planejamento
Duration	0
Pre Situation	
Post Situation	
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)

Type	
Priority	0
How to Perform	<ul style="list-style-type: none"> • Método de Gerenciar Fase de Planejamento
Sub Tasks	
Actions	<ul style="list-style-type: none"> • Formular Escopo do Projeto • Avaliar Soluções • Tomar Decisões de Compra e Construção • Planejar Iterações • Planejar Business Case

Descriptor for Task Gerenciar Fase de Planejamento

Action	
Number	1.2.1
Name	Avaliar Processo de Construção
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	

	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Processo de Construção

Action	
Number	1.2.2
Name	Avaliar Ambiente
Description	O gerente avalia o ambiente de desenvolvimento em vista de fazer melhorias
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Ambiente

Action	
Number	1.2.3
Name	Avaliar Suporte à Automação
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Suporte à Automação

Action	
Number	1.2.4
Name	Avaliar Milestones
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	

Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Milestones

Action	
Number	1.2.5
Name	Avaliar Arquitetura
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Arquitetura

Action	
Number	1.2.6
Name	Avaliar Componentes
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Componentes

Task	
Number	1.2
Name	Gerenciar Fase de Elaboração
Description	Gerente de projeto gerencia a fase de Elaboração
Duration	0
Pre Situation	
Post Situation	
Frequency	
Importance	

Interruptability	
Modality	
Occurrence	(1,n)
Type	
Priority	0
How to Perform	<ul style="list-style-type: none"> • Método de Gerenciar Fase de Elaboração
Sub Tasks	
Actions	<ul style="list-style-type: none"> • Avaliar Processo de Construção • Avaliar Ambiente • Avaliar Suporte à Automação • Avaliar Milestones • Avaliar Arquitetura • Avaliar Componentes

Descriptor for Task Gerenciar Fase de Elaboração

Action	
Number	1.3.1
Name	Gerenciar Recursos
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	

Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Gerenciar Recursos

Action	
Number	1.3.2
Name	Otimizar Processo
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Otimizar Processo

Action	
Number	1.3.3
Name	Avaliar Versões do Produto
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Versões do Produto

Task	
Number	1.3
Name	Gerenciar Fase de Construção
Description	Gerente de projeto gerencia a fase de Construção
Duration	0
Pre Situation	
Post Situation	
Frequency	
Importance	

Interruptability	
Modality	
Occurrence	(1,n)
Type	
Priority	0
How to Perform	<ul style="list-style-type: none"> • Método de Gerenciar Fase de Construção
Sub Tasks	
Actions	<ul style="list-style-type: none"> • Gerenciar Recursos • Otimizar Processo • Avaliar Versões do Produto

Descriptor for Task Gerenciar Fase de Construção

Action	
Number	1.4.1
Name	Sincronizar Versões
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	

	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Sincronizar Versões

Action	
Number	1.4.2
Name	Avaliar Versões
Description	
Duration	0
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Pre Situation	
Post Situation	
Priority	0
Agents	<ul style="list-style-type: none"> • Gerente de Projeto
Tools	<ul style="list-style-type: none"> • Ferramenta para gerência de processos

Descriptor for Action Avaliar Versões

Task	
Number	1.4
Name	Gerenciar Fase de Transição
Description	Gerente de projeto gerencia a fase de Transição
Duration	0
Pre Situation	
Post Situation	
Frequency	
Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Priority	0
How to Perform	<ul style="list-style-type: none"> • Método de Gerenciar Fase de Transição
Sub Tasks	
Actions	<ul style="list-style-type: none"> • Sincronizar Versões • Avaliar Versões

Descriptor for Task Gerenciar Fase de Transição

Task	
Number	1
Name	Gerenciar Processo de Software
Description	Gerente de processo gerencia o processo de desenvolvimento do software
Duration	0
Pre Situation	
Post Situation	
Frequency	

Importance	
Interruptability	
Modality	
Occurrence	(1,n)
Type	
Priority	0
How to Perform	<ul style="list-style-type: none"> • Método de Gerenciar Processo de Software
Sub Tasks	
Actions	

Descriptor for Task Gerenciar Processo de Software